א טיפער איבערבליק אויף אידיש – די פולע, לענגערע אויסגאבע

YouTube 7 ווידעאס: AI, מאשין-לערנען, AI אגענטן, Pixel 9a און Veo 3.1

טעקע 2 פון 3: טייל 4, ערשטע העלפט – ביז 26:32:13

אלעס וואס ווערט געזאגט און געוויזן אין די ווידעאס, כמעט שריט ביי שריט

ארום 83 שעה ווידעא

אקטאבער 2026

אינהאלט

טייל 4: דער גאנצער Agentic AI קורס 2027 – Simplilearn 5

וועגן דעם וויידעא 5

אריינפיר אין Agentic AI און די יסודות פון AI 7

פון LLMs צו אגענטן און פראקטישע ניצונג 12

גענעראטיווע AI און די קונסט פון פראמפטן 17

פראמפט-פארבעסערונג און AI פאר פארשונג און אינהאלט 22

פון רויע דאטא צו קלוגע איינזיכטן מיט AI 26

פון גענעראטיוון AI ביז אגענטיק AI 31

די שטאפלען פון AI און דער אנהייב פון גענעראטיוון AI 36

E·ChatGPT, DALL און GPT-3 אין דער פראקטיק 40

דזשענערעטיוו מאדעלן און דער ערשטער פראקטישער מאדעל 44

אנווענדונגען, א טשעטבאט, און די גרענעצן פון AI 49

חסרונות, צוקונפט, און ChatGPT פאר דאטא-אנאליז 54

דאטא-אנאליז און לערנען Python מיט ChatGPT 58

פון Python און Java מיט ChatGPT ביז LLM-ס 61

טראנספארמערס, LLM-ארכיטעקטור און א ערשטער פראיעקט 64

סוף פון ערשטן LLM און אריינפיר אין Transformer 68

טראנספארמערס, אטענשאן און גרויסע שפראך-מאדעלן 72

פעאיגקייטן, סארטן און די עפענע מאדעלן 77

אפן-סאורס מאדעלן: לייסענסעס, פראיעקטן און די שפּיץ־מאדעלן 82

אפן-סאורס מאדעלן, געצייג, קאסטן און ריזיקעס 86

קאנטעקסט-פענצטער און די יסודות פון פראמפט-אינזשעניריע 91

פראמפט-עלעמענטן אין פראקטיק און אנווענדונגען 96

פראמפט-מוסטערן און געווייַנטלעכע פראמפט-טעותים 99

פראמפט-טעותים און פארגעשריטענע טעכניקן 103

קאנטעקסט-אינזשענירינג קעגן "וייב-קאודינג" 107

פון פרעדיקציע ביז פראמפט-אינזשעניערינג 112

פון קלאסישן AI ביז גענעראטיוון AI 116

פון RNN ביז LLM: אטענשאן און טראנספארמערס 120

גענעראטיווע AI, זארגן, און גרויסע שפראך-מאדעלן 123

וועלכע AI געצייג צו קלייבן, און א רעזומע מיט ChatGPT 126

וואס איז א פראמפט און פון וואס באשטייט ער 130

פראמפט-פריימווערקס און פראקטישע איבונגען 133

מולטימאדאל AI און די געשיכטע פון LLMs 136

איבערבליק: LLMs, פראמפטן און מולטימאדאל 140

טאָקענס, קאוד-געווראַציע און וועקטאָרס פון GPT 143
שרייב-געזייג און דיזיין מיט Canva AI 146
ספעציאליזירטע GPT געזייג: דיזיין, פארשונג און לערנען 149
פראמפט-אינזשעניערינג: קלארע אנווייזונגען פאר AI 153
פארגעשריטענע פראמפט-טעכניקן: zero-shot, few-shot און chain of thought 157
פראמפט-מעטאדן און מולטימאדאל פראמפטינג 160
פון מולטימאדאל ChatGPT ביז LangChain 165
LangChain: מאדעלן, פראמפטס און די ערשטע קאוד 169
פראמפט-מוסטערן, אויסגאנג-פארזערס און זכרון 173
זכרון אין LangChain און די ערשטע טריט מיט chains 177
קייטן אין LangChain: פון איינפאכע ביז ראוטינג 181
פראגע-ענטפער סיסטעם, אפשאצונג, און אגענטן מיט LangChain 185
אגענטן מיט LangChain און א הקדמה צו LLM-טשעטבאטן 188
צווייַ סארטן טשעטבאטס און ווי מען בויט זיי 192
סענטימענט-אנאליז און א LangChain טשעטבאט מיט Streamlit 196
פארענדיקן דעם Streamlit טשעטבאט און אריינפירן LangChain 199
LangChain: אינסטאלאציע, Chains, פראמפטס און RAG 203
א RAG-אפליקאציע מיט LangChain און FAISS 207
ענדיגן דעם RAG פראיעקט, אגענטן און וועקטארן 210
וועקטאר-דאטאבאזעס: באדייטונג אנשטאט פונקט-וועדטער 213
וועקטאר-דאטאבאזעס, RAG און Chroma DB 218
Pinecone: וועקטאר-דאטאבעיס און סעמאנטישע זוך 221
Pinecone מיט FastAPI, און אריינפיר אין RAG 225
דאקומענט-לאדער און טעקסט-צעטיילן אין LangChain 229
שפּאַלטן טעקסט, embeddings און וועקטאר-סטארס 233
ריטריוועל אין RAG: פעלערן, MMR און מער 236
פיל-דאקומענט אגענט, די צוקונפט פון RAG, און Agentic RAG 240
אגענטישע RAG מיט LlamaIndex און דער Router Query Engine 244
דער Router Query Engine אין פראקטיק און Tool Calling 249
RAG מיט טול-רופן: טעאריע און א דעמא 252
נאדעס, וועקטאר-אינדעקס און אגענט-ריזאנירן 256
פונקציע-רופנדיקע אגענטן און קאמפליצירטע פראגעס 260
מולטי-דאקומענט אגענט און אריינפיר אין RAG 264
RAG-פראיעקט סטרוקטור, פיין-טיונינג און פראמפט-אינזשעניערינג 269
פראמפט-אינזשעניערינג און מולטימאדאל RAG 273
א מולטימאדעל RAG טשעטבאט און ווידעא-פאראַרבעטונג 278
וויִדעא-פראצעסירונג און וועקטאר-סטאָרס אין מולטימאדאל RAG 282
מולטימאדעל זוכן מיט LanceDB און LVLM מאדעלן 286

מולטימאדאל RAG מיט LLaVA און LangChain 290
וועב-קראולינג, RAG און Crawl4AI 294
Crawl4AI פאר RAG און וואס אי׳ז אן AI אגענט 298
ווי אזוי אגענטן ארבעטן, און דער ערשטער בויען מיט browser use 302
אינסטאלירן Web UI און לויפן א בראוזער-אגענט 305
ערשטע אגענטן מיט OpenAI, Gemini און דרויסנדיגע API 308
אגענט מיט לאנג-טערמיניגן זכרון און פראמפט-קייטן 312
קייטן, ראוטינג, פאראלעל-ארבעט און אן אינטערוויו-קאוטש 317
דער AI אינטערוויו-קאוטש: זכרון, ראוטער און הויפט-פראגראם 321
געזעראטיוע AI אגענטן: יסודות, ארבעט און ארכיטעקטור 325
אגענטיק AI: יסודות, באגרענעצונגען און צוקונפט 329
פריימווערקס, זכרון און פלאנירן בי״י אגענטן 334
פלאנירן, באנוץ-פעלער, אפשאצונג און די צוקונפט פון אגענטן 339
בויען אן AI אגענט פאר אפוינטמענטס מיט N8N 344
אויטאמאטישע באשטעטיגונג פאר אפוינטמענטס מיט n8n 348
סערווערלעס אגענטישע ארבעטס-פלוסן אין Amazon Bedrock 351
אגענט-אליאס, שמועס מיט׳ן אגענט און אקציעס 355
אגענט-טעסטן, גארדרעילס און קאוד-אינטערפרעטער אין Bedrock 358
אגענט מיט Code Interpreter און דאקומענטן 362
צוגעבן א וויסן-באזע און דער Bedrock קאנסאל 366
Bedrock קאנסאל און דער ערשטער n8n וואָרקפלאו 370
n8n: זכרון, געצייג, פלענער און וואס איז אגענטישע AI 375
אגענטן: בוי-שטיינער, ריזיקעס און דער ערשטער דעמא 379
פון סעטאפ-קאנטראל ביז וועב-אגענט 383
בריט דאטא און Xpander AI: דאטא און אגענטן 388
אגענט-באריכט און ChatGPT-ס זכרון 392
ChatGPT זכרון און AI-ארבעטס-אויטאמאציע 395
פינף אויטאמאציע-געצייג און א Zapier ביישפיל 399
Zapier-אגענטן, באשלוסן און Webhooks 403
OpenAI o1: א מאדעל וואס טראכט 406
סטרוקטורירטע פראמפטס און פלאנירן מיט o1 410
פלאנירן און אויספירן: סופליי-טשעין און קאוד-פארגלייך 415
קאוד-רעוויו און אנאליזירן בילדער מיט o1 419

טייל 4: דער גאנצער Agentic AI קורס 2027 – Simplilearn

נאמען: Agentic AI Full Course 2027 | Complete Agentic AI Course For Beginners | Simplilearn

קאנאל: Simplilearn

דאטום: 13 אקטאבער 2026

לענג: 42:37:52

לינק: https://youtu.be/36T5jhJLM8A

וועגן דעם וויידעא

וואס שטייט אין דער באשרייבונג

דער וויידעא הייסט "Agentic AI Full Course 2027", פונעם קאנאל Simplilearn. ער איז ארויפגעלייגט געווארן דעם 13טן אקטאבער 2026 און איז זייער לאנג: 42 שעה, 37 מינוט און 52 סעקונדעס. אויף דער יוטוב-בלאט זענען נישטא קיין איינגעטיילטע קאפיטלעך.

לויט דער באשרייבונג וועט מען אין דעם וויידעא לערנען ווי אזוי צו פלאנירן, בויען און ארויסשטעלן אינטעליגענטע AI אגענטן. די אגענטן קענען טראכטן, פלאנירן, ניצן געצייג און דורכפירן קאמפליצירטע אויפגאבעס כמעט אן מענטשליכע אריינמישונג. דער קורס איז געמאכט פאר אנהייבער. ער פירט פון די יסודות פון ביז סיסטעמען מיט עטליכע אגענטן (multi-agent), אויטאמאטיזירן ארבעט-פראצעסן (workflow automation), און אגענטישע AI אפליקאציעס וואס זענען גרייט פאר אמת'ע באניץ (production).

דער סדר הלימוד, ווי די באשרייבונג שילדערט אים:

- מען הייבט אן מיט די יסודות פון גענעראטיוע AI און גרויסע שפראך-מאדעלן (LLMs).
- דערנאך גייט מען איבער צו פראקטישע ארבעט מיט AI אגענטן, צוזאמבינדן געצייג (tool integration), RAG, זכרון פאר אגענטן (agent memory), און פריימווערקס וואס אָרגאניזירן און פירן די אגענטן (orchestration).
- צום סוף זאל מען פארשטיין ווי אזוי צו בויען א פולשטענדיגע אגענטישע AI סיסטעם, פון אנהייב ביז סוף, פאר אמת'ע געשעפט-באדערפענישן.

דער רוב פון דער באשרייבונג איז א רעקלאמע פאר א באצאלטע פראגראם מיטן נאמען "Applied Agentic AI: Systems, Design & Impact". דער קאנאל שרייבט איבער איר דאס פאלגנדע:

• א לערן-וועג וואס דעקט די יסודות פון AI, גענעראטיוו AI, ווי LLMs ארבעטן פון אינעווייניג, און סיסטעמען מיט עטליכע אגענטן. דערצו אגענט-פלאנירונג, פלאנירונג-פריימווערקס, און א סטראטעגיע ווי אזוי צו ברענגען א פראדוקט אויפן מארק (go-to-market).

• די פראגראם דויערט 10 וואכן, אויף א שטאפל פאר מענטשן וואס ארבעטן אין פאך.

• "AI Impact Studio" פון Simplilearn, וואס זאל העלפן נאכפאלגן נייע טרענדס, אויסגעפינען געלעגנהייטן פאר AI, אויטאמאטיזירן ארבעט און בויען AI לייזונגען.

• לימודים אין אגענטישע פריימווערקס, סיסטעמען מיט עטליכע אגענטן, RAG, MCP, פלאנירונג-סיסטעמען, אויטאמאטיזירן ארבעט-פראצעסן און נאך.

• די געצייג און פריימווערקס וואס ווערן דערמאנט: LangChain, AutoGen, CrewAI, n8n, Lovable, Miro, Figma, LangSmith, Jupyter.

• איבערחזרן די יסודות פון Python און ארבעט-פראצעסן וואס פאסן פאר AI, איידער מען הייבט אן די הויפט-קורסן.

• לעבעדיגע קלאסן אנליין, מיט שמועסן צווישן די תלמידים אויף Slack, באגלייטונג פון מענטארן (mentoring), און הילף פאר יעדע גרופע.

• א געמיינזאמער סערטיפיקאט פון Microsoft און Simplilearn ביים פארענדיגן, און "Microsoft Learn" באדזשעס אויפן Microsoft Learn פארטאל פאר די קורסן מיטן נאמען פון Microsoft.

• א סערוויס "JobAssist" פון Simplilearn, וואס זאל העלפן אז גרויסע פירמעס וואס זוכן ארבעטער זאלן באמערקן דעם תלמיד.

די פעאיגקייטן (skills) וואס די פראגראם זאגט אז זי לערנט: אויטאמאטיזירן ארבעט-פראצעסן, אגענטישע פריימווערקס, סיסטעמען מיט עטליכע אגענטן, אינטעליגענטע אויטאמאטיזאציע, go-to-market פאר אגענטישע AI, עטיק און דורכזיכטיגקייט, LLM Function Calling, פלאנירונג-סיסטעמען, פראמפט-אינזשעניריע (Prompt Engineering), UX און UI מיט אגענטישע AI, Copilots, MCP און RAG.

אויבן אין דער באשרייבונג רעקלאמירט דער קאנאל אויך פינף באצאלטע פראגראמען:

• Microsoft Applied Agentic AI: Systems Design & Impact: א קורס אין אגענטישע AI.

• AI Accelerator Program, From Prompts to Agentic Workflows: א פראגראם וואס פירט פון פראמפטס ביז אגענטישע ארבעט-פראצעסן.

• Microsoft Applied Generative AI and Agentic AI Specialization: א ספעציאליזאציע אין גענעראטיוו AI און אגענטישע AI.

• Virginia Tech Applied Agentic AI: Systems Design & Impact: א קורס אין אגענטישע AI אונטערן נאמען פון Virginia Tech.

• Michigan Engineering Generative AI Applications for Leaders: א קורס וועגן גענעראטיוו AI אפליקאציעס פאר פירער און מענעדזשערס.

לינקס און רעסורסן וואס ווערן דערמאנט

- Microsoft Applied Agentic AI: Systems Design & Impact, דער הויפט באצאלטער קורס. אויף דעם לינק ווייזט אויך דער "Enroll Now" (פארשרייבט זיך) צום סוף: https://www.simplilearn.com/agentic-ai-course-training
- AI Accelerator Program, From Prompts to Agentic Workflows: https://www.simplilearn.com/ai-accelerator-program
- Microsoft Applied Generative AI and Agentic AI Specialization: https://www.simplilearn.com/microsoft-applied-ai-course
- Virginia Tech Applied Agentic AI: Systems Design & Impact: https://www.simplilearn.com/vt-applied-agentic-ai-course
- Michigan Engineering Generative AI Applications for Leaders: https://www.simplilearn.com/generative-ai-for-business-transformation
- פינף "פארבונדענע ווידעאס" (Related Videos) פונעם קאנאל: https://youtu.be/fqk1l81Cw2M, https://youtu.be/13JrXL0c9Jw, https://youtu.be/DZL8JOQq9io, https://youtu.be/Nw0R2jP-A0g, https://youtu.be/QP9MYqoiZRw
- א הינווייז צו נאך AI ווידעאס פון Simplilearn, א רשימה מיטן נאמען "Artificial Intelligence [2027 Updated]". דער לינק איז נישט אויסגעשריבן אין דער באשרייבונג.
- פון סאציאלע קאנאלן וועלט דערמאנט נאר א לינק זיך צו אבאנירן (subscribe) אויפן יוטוב-קאנאל: https://bit.ly/2VT4WtH

אריינפיר אין Agentic AI און די יסודות פון AI (0:05)

וואס איז Agentic AI, און דער פלאן פונעם קורס (0:05)

דער קורס הייבט אן מיט א בילד: שטעלט אייך פאר אז מען גיט א AI א ציל אנשטאט א פראגע. אנשטאט נאר צו בעטן דעם AI ער זאל עפעס שרייבן, זאגט מען אים ער זאל אויספארשן א טעמע, געפינען די ריכטיגע אינפארמאציע, ניצן פארשידענע געצייג, מאכן באשלוסן, און אליין פארענדיגן די ארבעט. דאס, וועלט ערקלערט, איז דער יסוד-געדאנק פון Agentic AI.

עס וועלט ערקלערט אז AI באוועגט זיך שנעל פון סיסטעמען וואס גיבן נאר ענטפערס צו סיסטעמען וואס קענען טראכטן און באטראכטן, פלאנירן, ניצן געצייג, געדענקען אינפארמאציע, און טון אקציעס. וועל עס וויל פארשטיין וואוהין AI גייט וועט ער, פאר דעם וועלט אלץ וויכטיגער צו לערנען ווי AI אגענטן ארבעטן.

דער קורס פירט פון די יסודות פון קינסטלעכע אינטעליגענץ און גענעראטיווע AI ביז צו בויען, אפשאצן, אריסשטעלן (deployment) און פירן סיסטעמען פון AI אגענטן. דער פלאן איז צעטיילט אין צוואנציג מאדולן:

- מאדול 1: יסודות פון AI און גענעראטיווע AI
- מאדול 2: גרויסע שפראך-מאדעלן (Large Language Models)
- מאדול 3: Prompt Engineering
- מאדול 4: אנטוויקלען אפליקאציעס מיט LLM

- מאדול 5: Embeddings און וועקטאר-דאטאבאזעס
- מאדול 6: Retrieval Augmented Generation (RAG)
- מאדול 7: יסודות פון AI אגענטן
- מאדול 8: ארכיטעקטור פון Agentic AI
- מאדול 9: געצייג און Function Calling
- מאדול 10: זכרון און קאנטעקסט פון אגענטן
- מאדול 11: טראכטן, פלאנירן און ארבעטס-פלוסן (workflows)
- מאדול 12: פריימווערקס פאר Agentic AI
- מאדול 13: Model Context Protocol (MCP)
- מאדול 14: סיסטעמען מיט עטליכע אגענטן (Multi-Agent Systems)
- מאדול 15: אפשאצן אגענטן און אכטונג געבן אויף זיי (Evaluation and Observability)
- מאדול 16: פאראנטווארטלעכע און זיכערע Agentic AI
- מאדול 17: פראיעקטן פון AI אגענטן פון דער ווירקלעכער וועלט
- מאדול 18: אריסשטעלן (Agentic AI (Deployment
- מאדול 19: LLMOps און MLOps פאר Agentic AI
- מאדול 20: קאריערע און אינטערוויוס אין Agentic AI

ביים סוף פונעם קורס, ווערט געזאגט, וועט מען האבן א שטארקן יסוד אין Agentic AI, און פארשטיין ווי מאדערנע AI סיסטעמען גייען פון שפראך-מאדעלן און פראמפטס צו RAG, אגענטן, געצייג, זכרון, ארבעטס-פלוסן, Multi-Agent סיסטעמען, deployment און LLMOps.

(דא קומט א קורצע רעקלאמע פאר א פראגראם פון Microsoft און Simplilearn וועגן Applied Agentic AI, מיט א צעטיפיקאט; מיר לאזן עס אויס.)

א קליינער קוויז צום אנהייב (3:08)

פארן אנהייבן, ווערט געשטעלט א קוויז: וואס באשרייבט צום בעסטן א AI אגענט? די ברירות:

- A: א סיסטעם וואס קען פארשטיין א ציל, מאכן באשלוסן, ניצן געצייג, און טון אקציעס.
- B: א סיסטעם וואס קען נאר אריסגעבן טעקסט.
- C: א דאטאבאזע פאר אפהאלטן AI מאדעלן.
- D: א פראגראמירונגס-שפראך פאר בויען וועבזייטלעך.

דער עולם ווערט געבעטן צו שרייבן זייער ענטפער אין די קאמענטארן. דער ענטפער ווערט דא נישט געזאגט.

וואס איז AI באמת? (3:38)

איידער מען רעדט פון געצייג, אגענטן אדער טשעט-באטס, ערקלערט דער קורס די הויפט-זאך: וואס איז AI? קינסטלעכע אינטעליגענץ מיינט פשוט צו שאפן קאמפיוטער-סיסטעמען וואס קענען טון ארבעטן וואס פאדערן געווענליך מענטשלעך טראכטן. ביישפילן: פארשטיין א פראגע, דערקענען א פנים אויף א בילד, איבערזעצן

פון איין שפראך צו אן אנדערע, פארשלאגן וואס צו קוקן ווייטער, שרייבן אן אימעיל, שאפן א בילד, אדער העלפן לייזן א פראבלעם.

דער גרינגסטער אופן צו פארשטיין AI: עס איז נישט קיין כישוף און נישט קיין מענטשלעכער מוח. עס איז א סיסטעם וואס איז געטרייניירט געווארן אויף גרויסע מאסן דאטא, אזוי אז עס קען דערקענען מוסטערן (patterns) און געבן נוצליכע רעזולטאטן. ווען מען בעט א AI צו ערקלערן א טעמע, קוקט עס אויף דער בקשה, פארשטייט דעם מוסטער פון דער שפראך, און שאפט א ענטפער וואס פאסט צו דער פראגע.

בפשטות, AI איז ווי א זייער שנעלער געהילף וואס האט זיך אויסגעלערנט פון ביישפילן. עס ווייסט נישט זאכן אויפן אופן ווי א מענטש, אבער עס קען באארבעטן אינפארמאציע שנעל און ענטפערן אויף א נוצליכן אופן. דערפאר ווערט AI היינט געוצט אין זוך-מאשינען, טעלעפאנען, קארס, קונה-סערוויס, חינוך, געזונט, דיזיין, מארקעטינג, שרייבן קאוד, און אסאך אנדערע געביטן.

פארוואס AI איז היינט אזוי וויכטיג (4:38)

AI איז וויכטיג היינט ווייל עס איז שוין נישט באגרעניצט צו גרויסע טעכנאלאגיע-פירמעס אדער פארשונגס-לאבאראטאריעס; עס איז אריין אין דער געוועלטליכער טאג-טעגליכער ארבעט:

- סטודענטן ניצן AI צו פארשטיין טעמעס שנעלער.
- ארבעטנדע פראפעסיאנאלן ניצן AI צו שרייבן אימעילס, צוגרייטן באריכטן, צוזאמנעמען מיטינגס, שאפן פרעזענטאציעס, אנאליזירן דאטא, און לערנען נייע פעאיגקייטן.
- ביזנעסער ניצן AI צו פארבעסערן קונה-סערוויס, פארמינערן איבערגעחזרטע ארבעט, פארשטיין דאס פירונג פון קונים, און מאכן שנעלערע באשלוסן.
- שאפער (creators) ניצן AI צו ברענגען געדאנקען פאר ווידעאס, thumbnails, סקריפטן, קול-איבערלייגן, אנימאציעס, און אינהאלט פאר סאציאלע מעדיע.

די סיבה איז פשוט: מענטשן וואס ווייסן ווי צו ניצן AI ריכטיג, קענען שפארן צייט, פארבעסערן קוואליטעט, און ארבעטן שנעלער. אבער דער שליסל-ווארט איז "ריכטיג". AI איז נישט דא צו פארטרעטן דאס טראכטן, נאר צו שטיצן דאס טראכטן. א ביישפיל: אויב א מענטש פארנעמט פינף שעה צו שאפן דעם ערשטן אנטוווורף פון א פרעזענטאציע, קען AI העלפן שאפן א גראבע ווערסיע אין עטליכע מינוט; אבער מען דארף נאך אלץ איבערקוקן, פארבעסערן, און צופאסן עס צום עולם.

דערפאר ווערט AI א יסודותדיגע פעאיגקייט, פונקט ווי דער אינטערנעט, אימעיל אדער spreadsheets. די פראגע איז נישט נאר "וועט AI ווירקן אויף מיין פרנסה?", נאר די בעסערע פראגע איז: "ווי אזוי קען איך ניצן AI צו ווערן בעסער אין מיין ארבעט?"

AI וואס מיר ניצן שוין יעדן טאג (6:09)

מען קען מיינען אז AI איז א נייע זאך, אבער דער קורס ווייזט אז רוב מענטשן ניצן שוין AI יעדן טאג אן צו באמערקן:

- ווען YouTube פארשלאגט א וויידעא וואס מען וועט אפשר ליב האבן.
- ווען Netflix פארשלאגט א פראגראם.
- ווען Google Maps זאגט פאראויס טראפיק און גיט א שנעלערן וועג.

- וועלן די קאמערע פון טעלעפאן פארבעסערט אויטאמאטיש א בילד.
- וועלן Gmail פארשלאגט א זאץ בשעת מען שרייבט.
- וועלן שאפינג-אפליקאציעס פארשלאגן פראדוקטן לויט פריערדיגע זוכן.
- וועלן בענק דערקענען אומגעווענליכע טראנזאקציעס און ווארענען וועגן מעגליכן שווינדל.

די וויכטיגסטע נקודה: AI איז נישט נאר טשעט-באטס. AI איז א טייל פון ווי אפליקאציעס פארשטייען באניצער, זאגן פאראויס זייערע ברירות, און מאכן פערזענליכע איבערלעבונגען. למשל, וועלן מען סקראלט אויף Instagram אדער YouTube Shorts, שטודירט די פלאטפארמע וואס מען קוקט, וואס מען שפרינגט איבער, וואס מען האט ליב, און וואס האלט אפ דעם באניצער; און לויט דעם ווייזט זי מער אינהאלט וואס פאסט צו זיין פירונג. דאס איז AI וואס ארבעט שטילערהייט. AI איז נישט קיין ווייטער עתיד-געדאנק; עס געפינט זיך שוין אין אונזערע טעלעפאנען, אפליקאציעס, וועבזייטלעך, קארס, אפיסן, און אפילו אין שטוב.

AI, מאשין-לערנען און דיפ-לערנען: דער ביישפיל פון א עסן-צושטעל אפליקאציע (7:10)

דער קורס ערקלערט די דריי באגריפן מיט איין פשוטן ביישפיל: אן אפליקאציע פאר צושטעלן עסן.

- AI: וועלן מען עפנט די אפליקאציע, פארשלאגט זי רעסטאראנען, זאגט פאראויס ווי לאנג דער צושטעל וועט נעמען, פאלגט נאך די באשטעלונג, און ענטפערט אפילו אויף פראגעס. די גאנצע קלוגע איבערלעבונג איז AI.
- מאשין-לערנען: שטעלט אייך פאר אז די אפליקאציע באמערקט אז די געווענליכע באשטעלונג פונעם באניצער איז קאווע אין אוונט. זי לערנט פון דעם פריערדיגן פירונג און הייבט אן וועלן בעסערע פארשלאגן. דער סיסטעם לערנט פון דאטא אנשטאט צו פאלגן נאר פעסטע כללים.
- דיפ-לערנען: אויב די אפליקאציע האט א טשעט-באט, און מען שרייבט "פארשלאג מיר עפעס שארף אונטער 300 רופיס", פארשטייט זי דעם זאץ ווי עס דארף צו זיין. דיפ-לערנען איז נוצליך פאר מער קאמפליצירטע ארבעטן, ווי פארשטיין שפראך, בילדער, קול, און שמועסן.

דער חילוק איז פשוט: AI איז דער אלגעמיינער קלוגער סיסטעם; מאשין-לערנען העלפט אים לערנען פון דאטא; און דיפ-לערנען העלפט אים באהאנדלען מער קאמפליצירטע זאכן ווי טעקסט, בילדער און רייד.

געזעראטיוו AI (8:10)

געזעראטיוו AI איז דער סארט AI וואס קען שאפן נייעם אינהאלט: טעקסט, בילדער, אודיא, ווידעא, קאוד, צוזאמענפאסונגען, און אפילו געדאנקען און דיזיינס, לויט וואס מען בעט.

פריער האט מען אסאך AI סיסטעמען געזוצט פאר פאראויסזאגן אדער קלאסיפיצירן. למשל, זיי האבן געקענט זאגן צי אן אימעיל איז spam אדער נישט, אדער צי א קונה וועט אפשר קויפן א פראדוקט. געזעראטיוו AI גייט איין טריט ווייטער: עס קען פראדוצירן עפעס נייעס. בעט מען עס צו שרייבן א באשרייבונג פון א פראדוקט, שרייבט עס איינע; בעט מען א געדאנק פאר א לאגא, קען עס העלפן; בעט מען צוזאמנעמען א לאנגע PDF, קען עס עס פארוואנדלען אין פשוטע פונקטן; און בעט מען א קאנצעפט פאר א ווידעא, קען עס געבן א סקריפט, א פלאן פון די סצענעס, און וויזועלע געדאנקען. דערפאר איז עס געווארן פאפולער ביי סטודענטן, שאפער, מארקעטירער, אנטוויקלער, לערער, ביזנעס-גרופעס, און כמעט יעדע אינדוסטריע.

אבער איין זאך איז וויכטיג: געזעראטיוו AI שאפט לויט מוסטערן וואס עס האט שוין געלערנט. עס קען זיין זייער נוצליך, אבער עס קען אויך מאכן טעותים. דערפאר דארף מען עס באהאנדלען ווי א קלוגן געהילף, און

נישט ווי א לעצטע אויטאריטעט. עס קען העלפן לערנען שנעלער, אנהייבן שנעלער, טראכטן בעסער, און אויספארשן מער געדאנקען; אבער מען דארף נאך אלץ מענטשלעכן שכל-הישר.

וואס איז א LLM (9:11)

LLM מיינט Large Language Model, א גרויסער שפראך-מאדעל. עס קלינגט טעכניש, אבער דער טייטש איז פשוט: א LLM איז א AI סיסטעם וואס איז געטרעינירט צו פארשטיין און שאפן מענטשלעכע שפראך. עס קען לייענען טעקסט, פארשטיין א פראגע, פארזעצן א זאץ, צוזאמנעמען אינפארמאציע, שרייבן אינהאלט, ערקלערן באגריפן, איבערזעצן שפראכן, און אפילו העלפן מיט שרייבן קאוד. געצייג ווי ChatGPT, Claude, Gemini און אנדערע AI געהילפן ארבעטן אויף גרויסע שפראך-מאדעלן.

דער קורס טייטשט אויס יעדעס ווארט:

- "Large" (גרויס): דער מאדעל האט געלערנט פון א ריזיגער מאס טעקסט-דאטא.
- "Language" (שפראך): ער ארבעט הויפטזעכליך מיט ווערטער, זאצן, און באדייטונג.
- "Model" (מאדעל): ער איז א געטרעינירטער סיסטעם וואס קען מאכן פאראויסזאגונגען.

אין עיקר, זאגט א LLM פאראויס וועלכע ווערטער דארפן קומען ווייטער, לויט דער פראגע און דעם קאנטעקסט. אבער ווייל ער איז געטרעינירט געווארן אויף אזויפיל שפראך, קען דער רעזולטאט זיך פילן נאטירליך און נוצליך. למשל, שרייבט מען "ערקלער Cloud Computing ווי פאר אן אנהייבער", פארשטייט דער מאדעל דעם סטיל וואס מען וויל און שאפט א גרינגע ערקלערונג. זאגט מען "מאך דעם אימעיל מער פראפעסיאנעל", שרייבט ער עס איבער בעסער. LLMs זענען נוצליך ווייל שפראך איז א טייל פון כמעט יעדע ארבעט: אימעילס, דאקומענטן, באריכטן, קאמענטארן אין קאוד, מעסעדזשעס פון קונים, לערן-נאטיצן, סקריפטן, און פרעזענטאציעס.

ווי אזוי ארבעט א LLM, און פארוואס דער פראמפט איז וויכטיג (10:43)

דער קורס ערקלערט עס אן צו גיין צו טעכניש. שטעלט אייך פאר אז מען שרייבט דעם זאץ "The sky is" (דער הימל איז). רוב מענטשן וועלן טרעפן אז דאס ווייטערדיגע ווארט איז "blue" (בלוי), ווייל מען האט געזען דעם מוסטער אסאך מאל. א LLM ארבעט ענליך, נאר אויף א פיל גרעסערן מאסשטאב: ער קוקט אויף די ווערטער וואס מען גיט אים, פארשטייט דעם באגריף, און זאגט פאראויס וואס דארף קומען ווייטער. דאס טוט ער ווידער און ווידער ביז ער שאפט א פולן ענטפער.

בשעת דעם טריינירן לייענט דער מאדעל א זייער גרויסע מאס טעקסט און לערנט מוסטערן אין שפראך: וועלכע ווערטער קומען צוזאמען, ווי מען ענטפערט אויף פראגעס, ווי ערקלערונגען זענען געבויט, ווי מען שרייבט קאוד, ווי מען שאפט צוזאמענפאסונגען, און ווי פארשידענע טאנען קלינגען.

ווען מען גיט אים א פראמפט, זוכט דער מאדעל נישט אין זיין זכרון ווי א מענטש וואס לייענט א בוך; ער ניצט די מוסטערן וואס ער האט געלערנט ביים טריינירן כדי צו שאפן א ענטפער. דערפאר איז דער פראמפט אזוי וויכטיג: אן אומקלארער פראמפט גיט אן אומקלארן ענטפער, און א קלארער פראמפט גיט א בעסערן ענטפער. א ביישפיל: "דערצייל מיר וועגן מארקעטינג" איז צו בריט; אבער "ערקלער דיגיטאלע מארקעטינג פאר אן אנהייבער אין פינף פשוטע פונקטן מיט ביישפילן" גיט דעם מאדעל א קלארערע ריכטונג.

פון LLMs צו אגענטן און פראקטישע ניצונג (11:45)

א ווארענונג וועגן LLMs, און וואס איז אן AI אגענט (11:45)

דער קורס ענדיגט די באהאנדלונג פון LLMs מיט א וויכטיגע ווארענונג: א LLM קען קלינגען זייער זיכער אין זיך אפילו ווען עס איז אינגאנצן פאלש. דערפאר דארף מען אלעמאל נאכקוקן וויכטיגע פאקטן, בפרט ווען עס האנדלט זיך ווען געוצליכע, מעדיצינישע אדער פינאנציעלע ענינים, אדער ווען אינפארמאציע פון די לעצטע צייטן.

דערנאך גייט מען איבער צום נעקסטן טריט: צו פארשטיין וואס איז אן AI אגענט. דער סאמע פשוט'סטער חילוק ווערט אזוי ערקלערט: א געוועלכער AI chatbot ענטפערט אויף א פראגע, בשעת אן AI אגענט פרובירט אויסצופירן א ארבעט.

א ביישפיל: אויב מען בעט א chatbot "שרייב א בריוו צו מיין מענעדזשער", וועט ער אפשר צוגרייטן דעם טעקסט פון א אימעיל. אבער אויב מען זאגט צו אן AI אגענט "פלאן מיין מיטינג, קוק אריין אין מיין קאלענדאר, גרייט צו דעם אימעיל און מאך א סדר-היום (agenda)", קען ער דורכגיין עטליכע טריט כדי צו דערגרייכן דעם ציל.

אן AI אגענט קען פארשטיין א ארבעט, זי צעטיילן אויף קלענערע טריט, ניצן געצייג (tools), נאכקוקן אינפארמאציע, און ווייטער ארבעטן ביז ער קומט אן צו א רעזולטאט. מען קען עס פארגלייכן מיט ארבעט וואס מען גיט איבער פאר א געהילף (assistant): מען פרעגט נישט בלויז נאך איין ענטפער, נאר מען גיט אים א ציל.

ביישפילן פון וואס אן AI אגענט קען העלפן:

- פארשן א טעמע
- פארגלייכן פארשידענע מעגליכקייטן
- צוגרייטן א באריכט (report)
- דערהיינטיגן א spreadsheet
- ארדענען נאטיצן
- העלפן ביי קאסטומער-סערוויס

טייל אגענטן קענען זיך פארבינדן מיט אפליקאציעס, פיילס, קאלענדארן, בראוזערס און אפילו ביזנעס-געצייג. עס ווערט דערמאנט אז Anthropic באשרייבט Claude Cowork אלס א געצייג וואס קען אויספירן ארבעטן אויף א קאמפיוטער, מיט לאקאלע פיילס און אפליקאציעס, און צוריקגעבן א פארטיגע ארבעט. דאס ווייזט ווי AI געצייג גייען איבער פון בלויז ענטפערן אויף פראגעס צו העלפן ענדיגן ממש ארבעט.

דער הויפט-געדאנק איז פשוט: AI אגענטן זענען נוצליך ווען א ארבעט פאדערט מער ווי איין טריט. אבער זיי דארפן נאך אלץ השגחה פון א מענטש, ווייל זיי קענען פאלש פארשטיין אנווייזונגען, ניצן פאלשע אינפארמאציע, אדער אפילו טון שריט וואס דארפן ערשט באקומען א דערלויבענישׂ.

Generative AI קעגן AI אגענטן (13:18)

יעצט, ווען מען פארשטייט סיי generative AI און סיי AI אגענטן, פארגלייכט זיי דער קורס אויף א פשוטן אופן: generative AI שאפט אינהאלט; AI אגענטן פירן אויס ארבעטן.

Generative AI איז נוצליך וועז מען וויל א רעזולטאט ווי טעקסט, א בילד, ציטאטן, א קיצור, א געדאנק פאר א ווידעא, א אימעיל אדער א דיזיין-געדאנק. AI אגענטן, אנדערש, זענען נוצליך וועז מען וויל אז א גאנצער פראצעס זאל אויסגעפירט ווערן, בפרט וועז די ארבעט האט עטליכע טריט.

ביישפיל: אויב מען זאגט "שרייב א LinkedIn פאוסט וועגן data analytics", קען generative AI העלפן צוגרייטן דעם פאוסט. אבער אויב מען זאגט "פארש אויס די לעבעדיגע trending טעמעס אין data analytics, פארגלייך זיי, מאך א קאלענדאר, און שרייב דריי LinkedIn פאוסטן", דאן קלינגט עס מער ווי א ארבעט פאר אן AI אגענט.

אן אנדער ביישפיל: generative AI קען צוגרייטן א רייזע-פלאן (itinerary). אבער אן AI אגענט קען אפשר נאכקוקן די דאטעס, פארגלייכן האטעלן, לייענען רעצענזיעס, אפשאצן די קאסטן, און צוגרייטן א פולשטענדיגן פלאן.

דער חילוק איז נישט וועלכער פון זיי איז בעסער, נאר וועלכן מען דארף טאקע. אויב מען דארף אינהאלט, איז generative AI גענוג; אויב מען דארף אז א ארבעט זאל גענטון ווערן טריט נאך טריט, איז אן AI אגענט אסאך מער נוצליך. אין 2026 איז דער חילוק וויכטיג, ווייל אסאך געצייג ווערן מער אויסגעשטעלט אויף טון פעולות: זיי ענטפערן נישט בלויז אויף פראגעס, נאר העלפן באניצער ארבעטן אינעווייניג אין אפליקאציעס, דאקומענטן, קאלענדארן, בראוזערס און שפעריישע געצייג. AI קען העלפן, אבער די לעצטע החלטה דארף בלייבן ביים מענטש וואס ניצט עס.

וואו מען ניצט AI אין דער וירקליכקייט (14:19)

דער קורס רעכנט אויס פארשידענע פעלדער וואו AI ווערט שוין גענוצט:

- חינוך: AI העלפט תלמידים פארשטיין שווערע טעמעס, מאכן א קיצור פון נאטיצן, זיך איבן מיט פראגעס, און לערנען לויט זייער אייגענעם טעמפא.
- שאפן אינהאלט: AI העלפט YouTubers, מארקעטינג-לייט און סאשעל-מידיע טימס צו שרייבן סקריפטן, קעפלעך (captions), thumbnails, טיטלען, באשרייבונגען, און געדאנקען פאר ווידעאס.
- ביזנעס: AI העלפט טימס שרייבן אימעילס, צוגרייטן באריכטן, אנאליזירן קאסטומער-רעאקציעס, און אויטאמאטיזירן ארבעטן וואס חזרן זיך איבער.
- געזונטהייט: AI קען העלפן דאקטוירים איבערקוקן מעדיצינישע בילדער, ארדענען אינפארמאציע פון פאציענטן, און שטיצן א שנעלערע דיאגנאז; אבער די לעצטע החלטות מוזן געמאכט ווערן דורך מעדיצינישע פאכמענער.
- באנקען: AI העלפט כאפן שווינדל (fraud), אפשאצן ריזיקעס, און פארבעסערן קאסטומער-סערוויס.
- e-commerce: AI רעקאמענדירט פראדוקטן, זאגט פאראויס די נאכפראגע, און פאסט צו דאס איינקויפן פאר יעדן באזונדער.
- סאפטוועיר-אנטוויקלונג: AI העלפט פראגראמירער פארשטיין קאוד, געפינען טעותים (bugs), שאפן test cases, און שרייבן דאקומענטאציע.
- HR: AI קען העלפן דורכקוקן רעזומעס, צוגרייטן אינטערוויו-פראגעס, און פארבעסערן די שטיצע פאר ארבעטער.

• קאסטומער-סערוויס: AI chatbots ענטפערן אויף געוועהנליכע פראגעס, אזוי אז די מענטשליכע טימס קענען זיך קאנצענטרירן אויף קאמפליצירטערע ענינים.

אפילו אין לאנדווירטשאפט, לאגיסטיק, פאבריקאציע און טראנספארטאציע ניצט מען AI צו פאראויסזאגן פראבלעמען, פארבעסערן דאס פלאנירן, און פארמינערן פארלוסטן. דער אמת'ער וועלט פון AI, ווערט באטאנט, איז נישט דאס וואס עס קען שאפן אינהאלט, נאר דאס וואס עס העלפט מענטשן מאכן די ארבעט שנעלער, פשוט'ער און מער ארגאניזירט.

פאפולערע AI געצייג אין 2026 (15:50)

דערנאך ווערן דורכגענומען די געצייג וואס מענטשן רעדן וועגן און ניצן אין 2026:

• ChatGPT: ווייט פארשפרייט פאר לערנען, שרייבן, הילף מיט קאוד, אויסטראכטן געדאנקען (brainstorming), שטיצע ביי פארשונג און לייזן פראבלעמען. לויט OpenAI'ס release notes זענען דא פארשידענע מאדעל-אפציעס: Instant פאר טאג-טעגליכע פראגעס, Thinking פאר טיפערע באטראכטונג (reasoning), און Pro פאר מער אוואנסירטע reasoning ארבעטן.

• Gemini: דער AI געהילף פון Google, געבוצט פאר שרייבן, פלאנירן, brainstorming און טאג-טעגליכע הילף, בפרט ווייל עס איז ענג פארבונדן מיט דער Google סביבה (ecosystem).

• Claude: געבוצט מערסטנס פאר שרייבן, באטראכטונג, מאכן קיצורים פון לאנגע דאקומענטן, און ארבעט-ענינים. Anthropic באשרייבט Claude אלס אן AI געהילף פאר שמועסן און באארבעטן טעקסט.

• Microsoft Copilot: נוצליך פאר די וואס ארבעטן מיט Microsoft געצייג, ווייל עס העלפט מיט שרייבן, ענטפערן אויף פראגעס, און שאפן בילדער. Microsoft באשרייבט עס אלס א דיגיטאלער באגלייטער פאר PC, Mac, מאביל און נאך.

• Perplexity : נוצליך ווען מען וויל פארשונג אין ענטפער-פארעם מיט מקורות; עס באשרייבט זיך אלס אן AI-געטריבענער "ענטפער-מאשין" וואס קען געבן ענטפערס אין דער צייט פון עכטן (real-time).

• Canva AI : פאפולער ביי שאפער, מארקעטערס און תלמידים, ווייל עס העלפט שאפן דיזיינס, ענדערבארע layouts, בילדער, דאקומענטן און וויזועלן אינהאלט אינעווייניג אין Canva.

• Runway: א פאפולערער AI געצייג פאר שאפן ווידאס; זיין Gen-4.5 בלאט שטעלט צענטראל ווידאע-געענעראציע מיט שטארקע באווענגונג-קוואליטעט און וויזועלע קאנטראל.

די וויכטיגע זאך, באטאנט דער קורס, איז נישט צו ניצן יעדן געצייג; דער קלוגער וועג איז אויסצוקלייבן געצייג לויט דעם וואס מען דארף.

ווי אזוי צו רעדן מיט AI: פראמפטינג (17:22)

יעצט לערנט מען די פעאיקייט וואס מאכט די אלע געצייג טאקע נוצליך. פראמפטינג מיינט פשוט צו געבן קלארערע אנווייזונגען פאר א AI געצייג. א פראמפט איז די מעסעדזש וואס מען טייפט. אסאך אנהייבער מיינען אז AI גיט שוואכע ענטפערס, אבער אפטמאל איז דער אמת'ער פראבלעם אז דער פראמפט איז צו אומקלאר.

ביישפיל: אויב מען שרייבט "explain AI ", קען דער ענטפער זיין צו ברייט. אבער אויב מען שרייבט "ערקלער AI פאר אן אנהייבער אין פשוטע ענגליש, מיט דריי ביישפילן פונעם לעבן, און אן טעכנישע ווערטער", וועט דער ענטפער זיין אסאך בעסער.

א גוטער פראמפט אנטהאלט געווענליך פינף זאכן:

- די ארבעט (task): זאגט פאר AI וואס צו טון.
- דער הינטערגרונט (context): גיט פאר AI דעם רקע.
- דער עולם (audience): פאר וועמען דער אינהאלט איז.
- די פארעם (format): ווי אזוי צו פארשטעלן דעם ענטפער.
- דער טאן (tone): ווי אזוי עס זאל קלינגען.

א ביישפיל פון א גוטן פראמפט: "שאף א 60-סעקונדן YouTube סקריפט וואס ערקלערט AI אגענטן פאר אנהייבער; ניץ א פשוט ביישפיל, האלט עס שמועסדיג, און ענדיג מיט א שטארקער שלוס-שורה." דער פראמפט איז בעסער ווייל ער גיט א ריכטונג.

נאך א נוצליכער מעטאד איז צו בעטן AI פארבעסערן זיין אייגענעם רעזולטאט. נאכן באקומען א ענטפער קען מען זאגן: "מאך עס פשוט'ער", "לייג צו ביישפילן", "מאך עס קירצער", "מאך עס מער ציענדיג", אדער "שרייב עס איבער פאר אן עולם וואס איז נישט טעכניש". פראמפטינג, ענדיגט דער קורס, איז נישט וועגן ניצן שיינע ווערטער, נאר וועגן זיין קלאר: וואס בעסער די אנווייזונג, אלץ בעסער דער רעזולטאט.

מעלות און חסרונות פון AI (18:54)

דער קורס רעכנט אויס פינף הויפט-מעלות:

- שנעלקייט: AI קען שאפן ערשטע דראפטן, מאכן קיצורים פון לאנגן אינהאלט, ערקלערן שווערע טעמעס, און געבן געדאנקען גיך.
- פראדוקטיוויטעט: עס קען פארמינערן ארבעט וואס חזר'ט זיך, ווי שרייבן רוטינע אימעילס, פארמאטירן נאטיצן, שאפן אויסלייגן (outlines) און ארדענען אינפארמאציע.
- לערנען: אן אנהייבער קען בעטן AI ערקלערן טעמעס אין פשוטע שפראך, געבן ביישפילן, שאפן קוויזן, און פארפשוט'ן קאמפליצירטע באגריפן.
- שעפערישקייט: AI קען העלפן ברענגען געדאנקען פאר ווידעאס, דיזיינס, קאמפיינס, מעשיות, פרעזענטאציעס און ביזנעס-פלענער.
- שטיצע פאר החלטות: AI קען פארגלייכן מעגליכקייטן, אנטדעקן מוסטערן (patterns), און העלפן ארגאניזירן אינפארמאציע.

אבער AI האט אויך באגרענעצונגען. עס קען מאכן טעותים; געבן פארעלטערטע אדער אומפולשטענדיגע אינפארמאציע; קלינגען זיכער אפילו ווען דער ענטפער איז פאלש; פאלש פארשטיין דעם פראמפט; אדער שאפן אינהאלט וואס פילט זיך צו אלגעמיין אויב מען גיט נישט דעם ריכטיגן הינטערגרונט. עס קען אויך אפשפיגלען נטיות (bias) פון דער דאטא אויף וועלכער עס איז געטרייניירט געווארן.

נאך א באגרענעצונג איז פריוואטקייט: מען זאל נישט אריינלייגן סענסיטיוו פירמע-דאטא, פאסווערטער, פריוואטע אינפארמאציע פון קאסטומערס, אדער קאנפידענציעלע דאקומענטן אין עפנטליכע געצייג, סיידן די פירמע ערלויבט עס. דער בעסטער וועג צו ניצן AI איז צו באהאנדלען עס ווי א נוצליכער געהילף, און נישט ווי דער וואס מאכט די לעצטע החלטה: ניצן עס צו פארגיכערן די ארבעט, אבער אלעמאל איבערקוקן דעם רעזולטאט.

זיבן טעותים פון אנהייבער (20:26)

• ערשטער טעות: פרעגן זייער בריייטע פראגעס. אז דער פראמפט איז אומקלאר, וועט דער ענטפער געווענליך אויך זיין אומקלאר.

• צווייטער: איבערקאפירן דעם AI רעזולטאט אן עס נאכצוקוקן. דאס איז ריזיקאליש, ווייל AI קען טועה זיין, בפרט מיט פאקטן, נומערן, אקטועלע געשעענישן, געזעצליכע עצות, געזונט-אינפארמאציע און טעכנישע פרטים.

• דריטער: ניצן AI בלויז פאר קורצע וועגן. AI זאל נישט פארביטן דאס לערנען; וועט עס קאפירט אלעמאל ענטפערס, קען ענדיגן די ארבעט, אבער וועט נישט אויפבויען קיין פארשטאנד.

• פערטער: נישט געבן קיין הינטערגרונט. "שרייב א סקריפט" איז שוואך; אבער "שרייב א 60-סעקונדן YouTube Short סקריפט פאר אן אנהייבער וועגן AI אגענטן, מיטן ביישפיל פון עסן-צושטעלונג (food delivery), אין פשוטע שפראך" גיט בעסערע רעזולטאטן.

• פינפטער: נישט פארבעסערן דעם רעזולטאט. דער ערשטער ענטפער איז נישט אלעמאל דער בעסטער; מען קען בעטן AI איבערשרייבן, פארפשוט'ן, פארברייטערן, מאכן מער ציענדיג אדער טוישן דעם טאן.

• זעקסטער: טיילן פריוואטע אינפארמאציע אן אכטונג. אנהייבער לייגן אמאל אריין קאנפידענציעלע דאטא אן טראכטן וועגן פריוואטקייט.

• זיבעטער: ערווארטן אז AI זאל אלעס טון פערפעקט. AI איז נוצליך, אבער עס דארף מענטשליכע ריכטונג, איבערקוקן און שכל.

די ריכטיגע צוגאנג איז פשוט: AI קען העלפן גיין שנעלער, אבער מען בלייבט אליין פאראנטוואורטליך פארן לעצטן רעזולטאט.

וואס צו לערנען ווייטער (21:26)

נאכן דורכגיין די יסודות (AI, generative AI, LLMs אגענטן, געצייג, פראמפטינג, מעלות, חסרונות און טעותים) וועלט געפרעגט: וואס זאל מען לערנען דערנאך? דער קורס שטעלט פאר פינף זאכן:

• בעסער פראמפטינג: זיך איבן שרייבן קלארערע פראמפטס פאר לערנען, שרייבן, פארשונג, פרעזענטאציעס, אימעילס און לייזן פראבלעמען.

• נאכקוקן AI רעזולטאטן: נאכקוקן מקורות, פארגלייכן ענטפערס, און דערקענען שוואכע אדער אלגעמיינע ענטפערס.

• AI געצייג פאר דיין פעלד: א תלמיד – AI פאר לערנען און פראיעקטן; א מארקעטער – AI פאר אינהאלט, SEO און קאמפיינס; א פראגראמירער – AI קאוד-געהילפן און באזיס-אויטאמאציע; א דיזיינער – AI דיזיין-און בילד-געצייג; א ביזנעסמאן – AI פאר באריכטן, מיטינגס, spreadsheets און קאמוניקאציע.

• באזיס-אויטאמאציע: וועו מען פארשטייט פראמפטינג, קענען געצייג ווי AI אגענטן און workflow אויטאמאציע העלפן פארבינדן ארבעטן איינע מיט די אנדערע.

• פאראנטוואורטליכע ניצונג פון AI: פארשטיין פריוואטקייט, גענויקייט, נטיות, און פארוואס מענטשליכע איבערקוקונג איז נויטיג.

דער בעסטער וועג ווייטער צו גיין איז נישט צו לערנען אלעס אויף איינמאל, נאר אויסקלייבן איין ניצונג פונעם טאג-טעגליכן לעבן און דארט אנהייבן – למשל מאכן א קיצור פון נאטיצן, פארבעסערן אימעילס, שאפן א

אויסלייג פאר א פרעזענטאציע, פארשן א טעמע, אדער פלאנירן אינהאלט. ווען מען ווערט באקוועם, גייט מען איבער צו מער אוואנסירטע ניצונגען.

AI אין 2026, ענדיגט דער קורס, איז נישט בלויז א טעכנאלאגישע טעמע, נאר עס ווערט א פראקטישע פעאיקייט; די וואס לערנען זיך עס צו ניצן קלאר, זיכער און שעפעריש, און ניצן AI טאג-טעגליך, וועלן האבן א שטארקן מעלה. צום סוף ווערט אנגעזאגט וואס קומט: מען וועט אויספארשן דעם אנהייב און דעם עיקר פון דער איבערקערעניש – ווי generative AI איז אנדערש פון טראדיציאנעלן AI, און ווי עס לערנט זיך צו שאפן אנשטאט בלויז צו קלאסיפיצירן.

געזעראטיוע AI און די קונסט פון פראמפטן (23:31)

דער פלאן פונעם קורס: פון פארשטיין ביז שאפן (23:31)

דער קורס ערקלערט דעם וועג וואס דער לערנער וועט דורכגיין. מען הייבט אן מיט די גרויסע דורכברוכן אין דער געשיכטע פון AI, פון די GPT מאדעלן ביז די diffusion סיסטעמען, וואס האבן געבויט איר אויפשטייג. דערנאך קוקט מען אויף דער וועלט "דורך AI'ס אויגן". מען לערנט פארשטיין די כח פון גרויסע שפראך-מאדעלן און נויראל-נעצן, וועלכע לאזן מאשינען טראכטן, קלערן און שאפן ווי קיינמאל פריער.

דערנאך גייט מען איבער פון פארשטיין צו אינטעראקציע. מען לערנט "רעדן די שפראך פון AI" דורך פראמפטן. יעדער גוטער רעזולטאט הייבט זיך אן מיט א גוטער פראגע. יעדע פראגע קען מען אויסשטעלן אזוי אז זי זאל ארויסברענגען שעפעריש קייט, געניקייט און איינזיכט.

מען וועט לערנען ווי אזוי צו פלאנירן ווירקזאמע פראמפטן און זיי פארבעסערן דורך פראקטיק. מען וועט אליין זען ווי קליינע ענדערונגען אין די ווערטער קענען איבערדרייען דעם ענטפער פון AI. דא טרעפט זיך שעפעריש קייט מיט קאנטראל, און דער באניצער ווערט דער וואס פירט.

ווייטער ווערט געוויזן ווי געזעראטיוע AI ווערט א שטארקער מיטהעלפער ביים שאפן און לערנען. מען קען עס נוצן צו פארשן קאמפליצירטע טעמעס, צו מאכן א קיצור פון ריזיגע מאסן אינפארמאציע, און צו שאפן אינהאלט: פון ארטיקלען און בילדער ביז פולע פרעזענטאציעס. Gen AI ווערט א שותף אין דער ארבעט.

מען וועט אויך לערנען ווי אזוי צו נוצן AI געצייג מיט אחריות, און דערביי היטן אויף ארגינעלקייט, געניקייט און עטישע אויפמערקזאמקייט. דער קורס באטאנט אז AI פארביטט נישט די מענטשליכע שעפעריש קייט, נאר עס פארשטארקט זי.

צום סוף ברענגט מען אלעס צוזאמען און לערנט ווי אזוי צו ניצן געזעראטיוע AI אין דער אמת'ער וועלט:

- ווי AI פארוואנדלט דאטא אין "מעשיות" און העלפט געפינען דעם באדייט וואס איז באהאלטן אין צאלן.
- ווי מען ניצט עס צו ברעינסטארמען, צו מאכן חידושים, און צו פלאנירן נייע לייזונגען שנעלער און קליגער.
- ווי AI פלעכט זיך אריין אין ארבעט-פראצעסן, הייבט אויף די פראדוקטיוויטעט, און ברענגט נייע פארמען פון דערציילן, אנאליטיק און חידושים.

צום סוף קוקט מען אויף דער צוקונפט: agentic AI און אויטאנאמע סיסטעמען, וואס קענען קלערן, געדענקען און מיטארבעטן לעבן אונז. דער קורס פירט דורך יעדן שטאפל פון דעם וועג: פון לערנען וואס AI איז, ווייטער צו בעל-הבית'ישקייט ווי אזוי צו רעדן מיט איר, און ביז שאפן און אנווענדן מיט איר. דער גרונט-געדאנק איז אז

די צוקונפט פון אינטעליגענץ איז נישט מער מאשינען וואס פאלגן, נאר מאשינען וואס שאפן צוזאמען מיט'ן מענטש.

וואס איז געמעראטיוע AI? (25:35)

עס ווערט ערקלערט אז ביז נישט לאנג צוריק האט די פעאיקייט צו שאפן קונסט, מוזיק אדער מעשיות געהערט נאר צו מענטשן. היינט קענען אלגאריטמען שרייבן לידער, מאלן פארטרעטן וואס זעען אויס ווי לעבעדיג, קאמפאנירן מעלאדיעס, און אפילו פירן נאטירליכע שמועסן. דאס טוישט אונזער גאנצן באגריף פון שעפערישקייט.

מען וועט אויפדעקן ווי געמעראטיוע AI האט זיך אנטוויקלט פון פשוט'ע דערקענונג פון מוסטערן ביז בראך-ברעכנדיגע סיסטעמען וואס שאפן אין גרויסן פארמעס. די טעכנאלאגיעס לערנען זיך נישט נאר צו אנאליזירן, נאר אויך "זיך פארשטעלן" און שאפן נייעם אינהאלט. דאס רירט אן ביזנעס, דערציאונג, פארווייטלונג און דאס טאגטעגליכע לעבן.

Generative AI, אדער Gen AI, איז א באזונדערע צווייג פון AI. זי אנאליזירט נישט סתם אינפארמאציע, נאר זי שאפט פון איר נייע זאכן. זי קען שרייבן עסעיס, שאפן רעאליסטישע בילדער, מאכן ווידעאס, קאמפאנירן מוזיק, און אפילו העלפן פלאנירן נייע פראדוקטן. דער באניצער האט עס אפשר שוין געהאט אויסגעפרואווט אן צו וויסן. למשל, ווען מען נוצט ChatGPT צו העלפן שרייבן עפעס, אדער ווען אן אפליקאציע פארוואנדלט א selfie אין א דיגיטאלע געמעל דורך AI.

די פארמעלע דעפיניציע וואס דער קורס גיט: געמעראטיוע AI איז א צווייג פון AI וואס קאנצענטרירט זיך אויף שאפן נייעם אינהאלט אדער דאטא. דאס שטיצט זיך אויף מוסטערן און וויסן וואס זי האט זיך אויסגעלערנט פון עקזיסטירנדע דאטא.

דער אונטערשייד צו טראדיציאנעלע AI:

- טראדיציאנעלע AI טוט מערסטנס אנאליזירן, קלאסיפיצירן אדער פאראויסזאגן רעזולטאטן.
- געמעראטיוע AI קען שאפן ארגינעלע רעזולטאטן: טעקסט, בילדער, קול, ווידעא, אדער אפילו קאוד.

זי לערנט זיך פון א סך דאטא צו זען מוסטערן און באציאונגען. מיט דעם וויסן שאפט זי נייעם אינהאלט וואס מאכט סענס, פאסט צום קאנטעקסט, און פילט זיך ארגינעל. דער קורס ברענגט צוויי משלים. ערשטנס, א קאך לערנט דורך אומצאליגע רעצעפטן כדי אויסצוטראכטן א נייע מאכל. צווייטנס, א מוזיקאנט הערט א סך לידער כדי צו קאמפאנירן אן אייגנארטיגע מעלאדיע. אזוי אויך קען Gen AI שרייבן מעשיות, שאפן קונסט-ווערק, קאמפאנירן מוזיק, אדער מאכן רעאליסטישע בילדער, אלעס באזירט אויף דעם וואס זי האט זיך אויסגעלערנט. בקיצור: געמעראטיוע AI איז א סיסטעם וואס קען זיך פארשטעלן און שאפן, נישט נאר דערקענען אדער אנאליזירן.

דער "צויבער-ראבאט": כחות און געצייג (27:40)

ווי אזוי איז דאס מעגליך? דער קורס גיט א משל. מען זאל זיך פארשטעלן AI ווי א צויבער-ראבאט מיט א מוח וואס לערנט זיך פון אלעס וואס ער זעט און הערט. פונקט ווי מענטשן לערנען דורך ליינען ביכער, הערן מוזיק אדער קוקן וויידעאס, אזוי לערנט AI פון ריזיגע מאסן דאטא. עס באמערקט מוסטערן, כאפט באציאונגען, און געדענקט וואס פאסט צוזאמען.

דער ראבאט האט פארשידענע "סופער-כחות" פאר פארשידענע ארבעטן:

• טעקסט-כח: שרייבט מעשיות, אימעילס אדער אפילו קאוד. א ביישפיל איז ChatGPT. אזעלכע מאדעלן הייסן text generative models.

• בילד-כח: שאפט צייכענונגען, אילוסטראציעס אדער רעאליסטישע פאטאגראפיעס. ביישפילן זענען DALL-E און Midjourney. זיי הייסן image generation models.

• קול-כח: קאמפאנירט מוזיק, שטימען אדער קלאנג-עפעקטן. דאס זענען audio generation models.

• ווידעא-כח: מאכט אנימאציעס אדער אפילו קורצע פילמען. דאס זענען video generation models.

• קאוד-כח: העלפט פראגראמירער שרייבן אדער פארענדיגן קאוד שנעל. א ביישפיל איז GitHub Copilot. דאס זענען code generation models.

אבער כחות אליין זענען נישט גענוג. יעדער "העלד" דארף געצייג. די AI געצייג זענען די טעכנאלאגיעס וואס לאזן די כחות ארבעטן:

• נויראל-נעצן: ווי די מוח-צעלן פונעם ראבאט. זיי העלפן דערקענען מוסטערן און פארבינדן אידייען.

• גרויסע שפראך-מאדעלן (LLM): א שפראך-מוח וואס פארשטייט טעקסט און קאנטעקסט, און קען שרייבן קלאר און צוזאמענהענגיג.

• Generative Adversarial Networks (GANs): דאס "קינסטלערישע אויג" וואס העלפט שאפן רעאליסטישע אדער פאנטאזיע-בילדער.

• Variational Autoencoders (VAEs): העלפן AI באהאנדלען קאמפליצירטע סטרוקטורן און ארגאניזירן קאמפליצירטע אינפארמאציע.

• Foundation models: אלגעמיינע מאדעלן וואס מען קען טריינירן פאר א סך ארבעטן, פון טעקסט ביז בילד ביז דאטא-אנאליז.

וען AI שאפט א מעשה, א בילד, מוזיק, ווידעא אדער קאוד, ניצט זי איר כח צוזאמען מיט די געצייג. די כחות באשטימען וואס צו שאפן. די געצייג מאכן זיכער אז זי קען לערנען, פארשטיין, און ארויסגעבן עפעס ארגינעלס און באדייטפולס. בקיצור: גענעראטיווע AI איז ווי א צויבער-ראבאט-פריינד. ער לערנט פון דער וועלט, ניצט זיינע כחות, וועגדט אן זיינע געצייג, און ברענגט אידייען צום לעבן.

מעלות, סכנות און אנווענדונגען (30:43)

גענעראטיווע AI ברענגט ריזיגע געלעגנהייטן, אבער אויך ריזיקעס.

מעלות:

• עס שפארט צייט, הייבט אויף שעפערישקייט, און העלפט לייזן פראבלעמען שנעלער.

• ביזנעסער קענען שאפן באריכטן, מארקעטינג-אינהאלט, אדער אפילו פראדוקט-דיזיין אין מינוטן.

• סטודענטן און לערער באקומען הילף מיט פארשונג, לערנען, און פערזענליכע ערקלערונגען.

• קינסטלער, שרייבער און מוזיקאנטן קענען אויספרובירן אידייען וואס זיי וואלטן זיך קיינמאל נישט פארגעשטעלט.

חסרונות:

• AI איז נישט פערפעקט. מאל מאכט עס טעותים, גיט פאלשע אינפארמאציע, אדער שאפט אינהאלט וואס מאכט קיין סענס נישט.

- עס זענען דא עטישע זארגן: קאפירן אן אנדערנ'ס ארבעט, פארשפרייטן פאלשע אינפארמאציע, אדער אריינרייסן זיך אין פריוואטקייט.

דערפאר איז וויכטיג צו ניצן AI מיט אחריות: זיכער מאכן אז עס איז יושר'דיג, דורכזיכטיג, און האט רעספעקט פאר מענטשנ'ס רעכטן.

אין וועלכע פעלדער עס מאכט שוין א גרויסע השפעה:

- ביזנעס: עס העלפט פירמעס שרייבן אימעילס, שאפן רעקלאמעס, פלאנירן פראדוקטן, און אויטאמאטיזירן קונדן-סערוויס.
- דערציאונג: עס גיט פערזענליכע ערקלערונגען, שאפט פראקטישע איבונגען, און ערקלערט קאמפליצירטע טעמעס אויף א פשוט'ן אופן.
- פארוויילונג: עס העלפט שאפן מוזיק, פילמען, ווידעא-שפילן און קונסט-ווערק, און גיט שאפער נייע וועגן זיך אויסצודריקן.
- אנדערע פעלדער: געזונט, פינאנצן און פארשונג ניצן AI צו אנאליזירן דאטא, שאפן באריכטן, און העלפן אנטדעקן נייע לייזונגען.

געזעראטיוו AI איז מער נישט קיין פאנטאזיע פון דער צוקונפט. עס איז א געצייג וואס פורעמט שוין היינט ווי אזוי מיר ארבעטן, לערנען און שאפן.

Prompt Engineering: וואס איז א פראמפט? (32:45)

דער קורס פרעגט: צי איז מען שוין אמאל געווען פרוסטרירט פון אן אומגעריכטן ענטפער פון AI? אפשר ליגט דער סוד צו בעסערע ענטפערס נישט נאר אין דער מאשין, נאר אין ווי אזוי מען פרעגט. דא קומט אריין prompt engineering. דאס איז די קונסט און וויסנשאפט פון שרייבן אנווייזונגען וואס פירן AI צו געבויט, פאסיגע און שעפעריש רעזולטאטן. מען וועט לערנען נישט נאר וואס א פראמפט איז, נאר אויך ווי אזוי אים אויפצובויען. מען וועט אויספרובירן פארשידענע צוגאנגען, און פארגלייכן וועלכע רעזולטאטן זענען נישט גוט. דאס אלעס וועט מען זען אין פראקטיק אויף פלאטפארמעס ווי ChatGPT, GitHub Copilot און אנדערע Gen AI געצייג.

א פראמפט איז בעצם א סעט אנווייזונגען וואס מען גיט צו AI. מען זאל עס זיך פארשטעלן ווי מען פירט א תלמיד, א קינסטלער אדער א קאך צו שאפן פונקט וואס מען ווייל. וואס בעסער די אנווייזונג, אלץ בעסער דער רעזולטאט. יעדער פראמפט האט עטליכע הויפט-טיילן:

- אנווייזונג (instruction): וואס מען וויל אז AI זאל טון.
- קאנטעקסט (context): הינטערגרונט-אינפארמאציע וואס AI דארף האבן.
- אריינגאנג-דאטא (input data): די רויע מאטעריאלן וואס AI זאל ניצן.
- אויסגאנג-פארמאט (output format): ווי דער ענדגילטיגער רעזולטאט זאל אויסזען.

אן די טיילן איז עס ווי בעטן איינעם קאכן וועטשערע אן אים זאגן די אינגרעדיענטן אדער וועלכע סארט קיך מען וויל.

א ביישפיל: די מעשה פונעם ראבאט וואס מאלט (34:48)

פארוואס איז דער פראמפט אזוי וויכטיג? אויב די אנווייזונגען זענען אומקלאר, קען AI געבן א צופעליגן אדער אומנוציגן ענטפער. א קלארער, אויסגעשטעלטער פראמפט גיט פונקט וואס מען וויל. אין דער לעקציע ווערט געוויזן א פשוט'ער פראמפט: "write a story about a robot". דער רעזולטאט איז זייער אומבאשטימט. AI וווייסט נישט וועלכע סארט מעשה מען ווייל, ווי לאנג זי זאל זיין, און וועלכע פרטים אריינצונעמען.

דערנאך פארבעסערט מען דעם פראמפט: "write a story about a robot learning how to paint". מען לייגט צו באדינגונגען:

- זי זאל זיין פינף זאצן לאנג.
- מען זאל אריינלייגן א קאמישע שווערigקייט וואס דער ראבאט טרעפט ביים מאלן.
- זי זאל זיך ענדיגן מיט א פאזיטיוון אדער איבערראשנדן "טוויסט".

מיט מער קאנטעקסט און סטרוקטור פירט מען AI צו שאפן עפעס פיל מער ספעציפיש און אינטערעסאנט. מען זעט אז דער רעזולטאט איז פיל בעסער: א מעשה וואס איז אינטערעסאנט, קאמיש, און פונקט אין דער לענג וואס מען האט געוואלט.

ארבע סטראטעגיעס פאר בעסערע פראמפטן (36:23)

דער קורס זאגט אז prompt engineering באשטייט פון פרעגן בעסערע פראגעס. די סטראטעגיעס זענען:

- זיין ספעציפיש: אנשטאט "שרייב אן אימעיל", זאל מען זאגן "שרייב א פראפעסיאנעלן אימעיל צו א קליענט וואס ערקלערט א פארשפעטיגטן שיקונג".
- באשטימען א ראלע: מען בעט AI זיך אויפצופירן ווי א לערער, קאוטש, דעוועלאפער אדער דיזיינער. למשל: "דו ביסט א מארקעטינג-מומחה. שרייב א כאפנדיגן סלאגאן פאר א וועגאנישן snack-בראנד."
- ניצן few-shot ביישפילן: מען ווייזט AI עטליכע ביישפילן פון וואס מען וויל איידער מען בעט עפעס נייעס. למשל: א fast food לאגא קען מען מאכן מיט א רויט-געלן בורגער-אייקאן. א טעק-פירמע לאגא קען האבן א בלוי-וווייסן אבסטראקטן דיזיין. דערנאך בעט מען: "יעצט שאף א לאגא פאר א פיטנעס-בראנד."
- system messages: מען גיט הויכע-שטאפל אנווייזונגען וואס פירן דאס אויפפירונג פון AI . למשל אין ChatGPT קען מען געבן: "always respond in a friendly tone".

דעמא: א פילם-chatbot אן קאוד (37:25)

צום סוף וועגדט מען אן וואס מען האט געלערנט, און מען בויט א פשוט'ן chatbot וואס ענטפערט פראגעס וועגן פילמען. מען קען ניצן ChatGPT אדער סיי וועלכן אנדערן LLM מאדעל. די טריט:

- ראלע: "you are a movie guide".
- אנווייזונג: גיב א סך-הכל פון דריי זאצן, און איין אינטערעסאנטע פאקט.
- קאנטעקסט: וואס דער באניצער קען פרעגן.
- אויסגאנג-פארמאט: "keep it concise and engaging".

מען זעט אז ChatGPT האט אפדעיטעד זיין זכרון (memory) און לאדנט איין פרעגן וועגן סיי וועלכן פילם. עס האט זיך אויטאמאטיש פארוואנדלט אין א chatbot וואס פירט זיך אויף ווי א פילם-גייד. מען פרעגט: "can you describe the movie Inception?" דער ענטפער גיט א קורצע ערקלערונג פונעם פילם פון 2010.

דערצו גיט עס אן אינטערעסאנטע פאקט: דער רעזשיסאר Christopher Nolan האט ביי דער סצענע פון דער "זיך-פאלדנדיגער שטאט" געבויט א פראקטישן סעט וואס האט געלאזט די געביידעס זיך בייגן. ער האט אויסגעמישט פראקטישע עפעקטן מיט CGI, צו שאפן א חלום-ארטיגע איבערלעבונג. דער קורס שליסט אז אן שום קאוד איז דער chatbot גרייט צום ניצן. דורך א קלארן פראמפט האט GPT פארשטאנען זיין אויפגאבע.

פראמפט-פארבעסערונג און AI פאר פארשונג און אינהאלט (39:58)

פראמפטן איז א פראצעס פון פרובירן און פארבעסערן (39:58)

דער קורס הייבט אן מיט א וויכטיגער באמערקונג: קיינער באקומט נישט פערפעקטע רעזולטאטן ביים ערשטן פרואוו. אמאל פארשטייט דער AI נישט וואס מען מיינט, אמאל גיט ער אן אומפולשטענדיגן ענטפער, און אמאל גייט ער אינגאנצן אוועק פון דער טעמע. דאס איז גאנץ נארמאל, ווייל ארבעטן מיט AI איז אן "איטעראטיוווער" פראצעס, א פראצעס פון חזר'ן און פארבעסערן. מען דארף פרובירן פארשידענע ווערסיעס פונעם פראמפט, טוישן די ווערטער, צולייגן ביישפילן, און זען ווי יעדע ענדערונג ווירקט אויפן רעזולטאט. אזוי לערנט מען זיך וואס מאכט דעם AI "טראכטן" בעסער.

מען זעט א ביישפיל פון א גאר אומקלארן פראמפט: "write about climate change", שרייב ווען קלימאט-ענדערונג. מען שיקט עס צו ChatGPT, און דער ענטפער איז נישט פאלש, אבער ער איז אויך נישט פאקוסירט. עס איז נישטא קיין קלארער ציל און קיין פארמאט. דאס פאסירט ווען דער פראמפט האט נישט קיין ריכטונג. א צווייטער ביישפיל: "explain AI". דא באקומט מען אויך אן אומקלארן ענטפער, ווייל עס זענען נישטא קיין אנווייזונגען און קיין קאנטעקסט. וועו א פראמפט איז צו אלגעמיין, ווייסט דער AI נישט ווי־פיל פרטים צו געבן און אין וועלכן טאן צו רעדן. מען ווייסט נישט צי דאס איז פאר א קינד, פאר א פארשער אדער פאר א ביזנעס-עולם.

צולייגן קאנטעקסט, סטרוקטור און פארמאט (41:01)

דערנאך ווערט געוויזן ווי מען מאכט דעם פראמפט קלארער. אנשטאט "שרייב ווען קלימאט-ענדערונג" שרייבט מען: "explain climate change for a 10-year-old student using simple language and one example", דאס הייסט ערקלער קלימאט-ענדערונג פאר א צען-יעריגן תלמיד, מיט פשוטע שפראך און איין ביישפיל. דא האט מען צוגעלייגט קאנטעקסט: ווער דער עולם איז און אויף וואספארא שפראך-שטאפל מען זאל רעדן. דער ענטפער קומט אויס קורץ און קלאר.

דער קומענדיגער שריט איז צולייגן סטרוקטור. מען בעט די זעלבע ערקלערונג פאר א צען-יעריגן תלמיד, אבער אין דריי קורצע פאראגראפן:

- דער ערשטער פאראגראף: וואס איז עס?
- דער צווייטער: וואס פאראורזאכט עס?
- דער דריטער: וואס קען מען דערוועגן טון?

אזא פראמפט ארגאניזירט דעם ענטפער אויף א לייענבארערן אופן, און דער רעזולטאט קומט אויס אויסגעשטעלט אין דריי טיילן.

א דריטער אופן איז אנצוגעבן דעם פארמאט פונעם ענטפער. מען בעט: ערקלער קלימאט-ענדערונג פאר א צען-יעריגן תלמיד אין "bullet points", אין פונקטן, מיט איין עמאדזשי ביים אנהייב פון יעדן פונקט. אזוי זאגט

מען דעם AI פונקטליך ווי ער זאל פרעזענטירן דעם ענטפער: פונקטן, עמאדזשיס און דער טאן. מען זעט ווי גענוי ער פאלגט די אנווייזונגען, און דער רעזולטאט איז אויסגעשטעלט און אינטערעסאנט. וועו מען פארגלייכט די לעצטע וועו ערסיע מיט דעם ערשטן ענטפער, זעט מען קלאר דעם חילוק: דער רעזולטאט איז קלארער, בעסער סטרוקטורירט און מער ציענדיג.

דריי געוועלכע פראבלעמען אוו ווי מען פארריכט זיי (44:07)

דער קורס ערקלערט אז אמאל, אפילו וועו דער פראמפט איז קלאר, מאכט דער AI נאך אלץ טעותים. עס ווערן געוויזן דריי פאלן:

- דער ענטפער איז צו לאנג. למשל דער פראמפט "explain Python function" איז קלאר, אבער דער ענטפער איז צו לאנג אדורכצוגיין. אין אזא פאל לייגט מען צו "keep it under four sentences", נישט מער ווי פיר זאצן, און מען באקומט א קורצערע וועו ערסיע. אין דער דעמא באקומט מען טאקע געוי פיר זאצן וואס פאסן צוזאמען וואס א פונקציע אין Python טוט.
- דער ענטפער איז אוועק פון דער טעמע אדער נישט שייך. דער פראמפט "write a story about data" ברענגט ארויס צופעליגע טעמעס וואס האבן נישט צו טון מיט דאטא. דער תיקון: "100-word story about a data scientist discovering a pattern", א מעשה פון הונדערט ווערטער וועגן א דאטא-סייענטיסט וואס אנטדעקט א מוסטער. אזוי ווערט די טעמע געהאלטן אויפן פלאץ.
- עס פעלן פרטים אדער שעפערישקייט. דער פראמפט "describe a city" ברענגט א גאר פלאכע באשרייבונג. מען טוישט אים אויף "describe a futuristic city in 2050 focusing on technology and nature", באשרייב א צוקונפטיגע שטאט אין 2050 מיט פאקוס אויף טעכנאלאגיע און נאטור, און מען באקומט א סך מער שעפעריש ענטפער ווי א שטאט אין 2050 וועט אויסזען, אינגאנצן פון דער פאנטאזיע פונעם AI.

AI פאר פארשונג, לערנען און אינהאלט: די פלאן פונעם מאדול (47:22)

א נייער מאדול הייבט זיך אן. אין א וועלט וואס לויפט שנעלער ווי אמאל איז די פראגע ווי מען קען לערנען, פארשן און שאפן אינהאלט אויף אן עפעקטיוון אופן. AI קען זיין א שותף וואס העלפט פינקטליך אנטדעקן אינפארמאציע, פארשטארקן שעפעריישקייט און שאפן ארבעט פון הויכער קוואליטעט אין ווייניגער צייט, און דערביי היטן אויף ארגינעלקייט און עטיק. דאס איז וואס גענעראטיוועו AI קען טון וועו מען ניצט אים פאר פארשונג, לערנען און אינהאלט. דער מאדול ווייזט ווי צו ניצן AI נישט נאר אלס צ'עטבאט, נאר אלס אן אמת'ן פארשונג- און שאפונג-שותף. די טעמעס:

- AI-פארשטארקטע פארשונג: ווי מען ניצט גענעראטיוועו AI צו זוכן, פארקירצן און ארגאניזירן פארשונג-מאטעריאל אדער דאטא.
- AI פארן לערנען: ווי AI קען זיין א לערן-שותף, וואס ערקלערט טעמעס, ענטפערט שאלות און שאפט פערזענליכע לערן-מאטעריאלן.
- AI פאר אינהאלט-שאפונג: ווי מען שאפט ארטיקלען, סקריפטן, בילדער און פרעזענטאציעס מיט AI-געצייג.
- פארגעשריטענע פראמפטן פאר אינהאלט: ווי מען שטעלט צוזאם פראמפטן פאר ספעציפישע געברויכן ווי SEO, סאציאלע מעדיע אדער אקאדעמישע שרייבונג.

- פארגלייכן געצייג: א קורצער בליק אויף עטליכע פון די בעסטע AI -געצייג פאר פארשונג און שאפונג, ווי ChatGPT, Bing, Copilot, Gemini און אנדערע.

ChatGPT קעגן Perplexity, און פארקירצן פארשונג-ארבעטן (48:25)

מען פארגלייכט צוויי LLM-מאדעלן: ChatGPT און Perplexity. צו ChatGPT גיט מען דעם פראמפט: "summarize recent research on LLM memory architectures", פארקירץ נייע פארשונג וועגן זכרון-ארכיטעקטורן ביי LLM-ס. מען באקומט א קלארע איבערזיכט וואס דעקט טעכניקן ווי retrieval-augmented generation (RAG), episodic memory און vector database. דערנאך גיט מען דעם זעלבן פראמפט צו Perplexity Pro. עס גיט דעם זעלבן אינהאלט, אבער מיט לעבעדיגע ציטאטן פון וועב, וואס מאכט עס מער פארלעסליך פאר אקאדעמישע רעפערענצן.

ווען מען האט שוין געפונען פארשונג-ארבעטן, איז די קומענדיגע שווערigקייט זיי צו פארקירצן און צוזאמצושטעלן. עס ווערט ערקלערט אז עס זענען דא צוויי סארטן AI-פארקירצונג:

- extractive summarization: קלייבט אויס וויכטיגע פראזעס אדער זאצן גלייך פונעם טעקסט.
- abstractive summarization: שרייבט איבער דעם אינהאלט מיט אייגענע ווערטער, און היט אפ דעם באדייט.

א ביישפיל: מען נעמט דעם אבסטראקט פון א פארשונג-ארבעט, לייגט אים אריין אין צ'עט און בעט: פארקירץ די ארבעט וועגן transformer efficiency אין פינף פונקטן. מען באקומט א קורצע זאמלונג מיט די מעטאדן, די רעזולטאטן און די באגרעניצונגען. דערנאך בעט מען: שרייב עס איבער אין א סטרוקטורירטן נאטיץ-פארמאט מיט טיילן: ציל, מעטאד, רעזולטאטן און באגרעניצונגען. אזוי ווערט אן אומסטרוקטורירטער טעקסט פארוואנדלט אין ניצבארע פארשונג-נאטיצן, גוט פאר אקאדעמישע אדער פראפעסיאנעלע דאקומענטאציע.

ווי קענען LLM-מאדעלן דאס טון? דער קורס ערקלערט אז זיי פארקירצן מיט'ן "attention mechanism", וואס דערקענט די טאקענס אינעם טעקסט וואס זענען דאס מערסטע שייך צום קאנטעקסט. אזוי פארשטייען זיי וועלכער טייל איז דער וויכטיגסטער.

ארגאניזירן וויסן: טאבעלעס און געצייג (51:34)

פארשונג איז נישט נאר לייענען; עס איז צוזאמבינדן געדאנקען. AI קען העלפן ארגאניזירן וויסן אין טעמעס, טאבעלעס און באגריף-מאפעס. א ביישפיל: מען האט פינף פארשונג-ארבעטן וועגן AI אין חינוך. מען לאדט ארויף אלע פינף צו ChatGPT און גיט דעם פראמפט: שאף א טאבעלע וואס פארקירצט יעדע ארבעט, מיט די קאלאמען: מחבר, טעמע, הויפט-איינזיכט און פארשונג-לוקע (research gap). דער מאדעל שטעלט גלייך צוזאם א סטרוקטורירטן פארגלייך.

עטליכע געצייג וואס מען קען ניצן:

- Notion AI: פארוואנדלט נאטיצן אין ארגאניזירטע דאטא-באזעס.
- Scholarcy: פארקירצט אויטאמאטיש PDF-טעקעס און באצייכנט וויכטיגע טיילן.

מען קען אויך ווידער ניצן פריערדיגע נאטיצן כדי צו האלטן א המשך. AI טוט נישט נאר אפהיטן דאטא; עס העלפט אין סטרוקטורירן וויסן, א וויכטיגע פעאיגקייט סיי אין אקאדעמישער און סיי אין פראפעסיאנעלער פארשונג.

שרייבן טעקסט און שאפן בילדער (53:10)

LLM-ס זענען גאר וועגדעוודיג אין שאפן געשריבענעם אינהאלט, צי עס איז אן אימעיל, א בלאג-ארטיקל, אן עסיי אדער א ווידעא-סקריפט. די מאדעלן פארשטייען קאנטעקסט, טאן און עולם, און מען קען דעריבער גרינג צופאסן דעם רעזולטאט. א ביישפיל: "write a 200-word blog on AI in education in a professional tone", א בלאג פון 200 ווערטער וועגן AI אין חינוך אין א פראפעסיאנעלן טאן. דער רעזולטאט קומט גלייך. כדי עס ווייטער צו פארפיינערן, בעט מען: מאך עס מער איבערצייגנדיג, לייג צו א "call to action" (אן אויפרוף צו טון עפעס), און שטעל עס אויס אין markdown. דער LLM טוישט דעם ווערטער-אוצר, דעם טאן און די סטרוקטור לויטן פראמפט.

דער קומענדיגער טייל איז וויזועלע דערציילונג, נאך א געביט וואס AI האט רעוואלוציאנירט. טעקסט-צו-בילד מאדעלן ווי Midjourney, DALL-E און Adobe Firefly קענען פארוואנדלען א טעקסט-פראמפט אין רעאליסטישע אדער קינסטלערישע בילדער. ChatGPT האט DALL-E איינגעבויט פאר שאפן בילדער. מען בעט: שאף א בילד פון א צוקונפטיגן קלאסצימער וואו AI-לערער העלפן תלמידים, מיט דיגיטאלע האלאגראמען אויף די וועגט, און מיט "cinematic lighting", קינא-ארטיגע באלייכטונג. אין עטליכע סעקונדעס שאפט דער AI אויסערגעווענליכע בילדער. מען קען קאנטראלירן דורכן פראמפט:

- דעם סטיל: רעאליסטיש, anime אדער) watercolorוואסערפארב).
- די באלייכטונג: cinematic אדער) soft lightווייכע ליכט).
- די פערספעקטיוו: א בליק פון אויבן (top view) אדער א נאענטער בליק (close-up).

פרעזענטאציעס מיט Canva AI (56:21)

דער קורס רעדט וועגן מולטי-מאדאלע געצייג ווי Canva AI, Tome, Gamma און Slides AI. די פלאטפארמעס פאראיינגן טעקסט, בילדער און דיזיין, און שאפן אויטאמאטיש א גאנצע פרעזענטאציע פון איין פראמפט. אין דער דעמא גיט מען Canva AI א פשוטן פראמפט: שאף א פרעזענטאציע פון פינף סליידס וועגן "the future of AI in healthcare", די צוקונפט פון AI אין געזונטהייט. Canva פרעגט וועלכער סטיל; מען קלייבט "minimalistic". דערנאך ווייזט עס דעם פלאן פון דער פרעזענטאציע, און מען קען דא מאכן ענדערונגען כדי מען זאל שפעטער נישט האבן קיין פראבלעמען. אויב אלעס זעט אויס גוט, דריקט מען "generate design". אין עטליכע סעקונדעס שאפט דער AI א טיטל-סלייד, הויפט-איינזיכטן, דאטא-טשארטס און בילדער וואס פאסן צו דער טעמע. א פרעזענטאציע וואס וואלט אמאל גענומען שעהן ארבעט, שעהן שאפונג און שעהן געדאנקען, האט גענומען בלויז עטליכע מינוט.

פראמפטן פאר פארשידענע פארמאטן (58:25)

די עפעקטיווקייט פון אייער AI הענגט אפ אין ווי גוט מען שטעלט צוזאם די פראמפטן, און פארשידענע פארמאטן פארלאנגען פארשידענע סטילן:

- SEO: "write an SEO blog on AI tools for students" מיט די שליסל-וועטער, AI learning assistant, productivity און study tools. אזוי גרינג שאפט מען א בלאג.
- סאציאלע מעדיע: "create three catchy LinkedIn posts about AI in content creation". מען באקומט דריי פאסטס וואס מען קען גלייך קאפירן אריין אין LinkedIn.
- אקאדעמיש: מען לייגט צו א דאקומענט און בעט: "summarize this paper in APA format with in-text citations", פארקירץ די ארבעט אין APA-פארמאט מיט ציטאטן אינעם טעקסט.

- סקריפט: "write a YouTube script explaining ethics in generative AI in an engaging tone". מען באקומט א גאנצן YouTube-סקריפט וואס מען קען גלייך לייענען, און די ארבעט איז פארטיג אין עטליכע סעקונדעס.

דער כלל: שטענדיג אנגעבן דעם עולם, די לענג און דעם פארמאט. דאס גיט דעם מאדעל א קלארע ריכטונג און ברענגט די רעזולטאטן פון "גוט" צו "אויסגעצייכנט".

עס ווערן אויך דערמאנט פארגעשריטענע פראמפט-מעטאדן:

- chain-of-thought prompting: בעטן דעם מאדעל ער זאל טראכטן שריט נאך שריט.
- role prompting: אים געבן א ראלע, למשל "act as a research assistant summarizing a scientific paper".
- output format control: בעטן די רעזולטאטן אין א markdown-טאבעלע אדער אין JSON-פארמאט.

די מעטאדן מאכן די רעזולטאטן סטרוקטורירט, ניצבאר אויף ווייטער, און עקביות'דיג.

א פראקטישע אויפגאבע: נייע טרענדס אין געזונטהייט אין אינדיע (1:02:06)

צום סוף שטעלט מען צוזאם אלעס וואס מען האט געלערנט אין א פראקטישער אויפגאבע: אויסגעפינען נייע טרענדס אין געזונטהייט-וועזן אין אינדיע, מיט ChatGPT און Perplexity. ערשט קלייבט מען Perplexity, ווייל זיין AI איז פארבונדן מיט'ן לעבעדיגן וועב, און עס איז דעריבער גוט פאר אקטועלע טרענדס. מען גיט דעם פראמפט: "what are the latest emerging trends in India's healthcare sector from 2023 to 2025". מען באקומט לעבעדיגע נייעס, איינזיכטן און באריכטן פון WHO, NITI Aayog און דעם אינדישן געזונטהייט-מיניסטעריום, צוזאמען מיט ציטאטן פון פארלעסליכע מקורות. דאס זענען די אלגעמיינע טרענדס וואס מען האט ארויסבאקומען.

פון רויע דאטא צו קלוגע איינזיכטן מיט AI (1:03:42)

טיפער פארשן מיט Perplexity: געזונט-טרענדס אין אינדיע (1:03:42)

דער קורס פירט ווייטער די דעמאנסטראציע ווי מען פארשט א טעמע מיט Perplexity. כדי צו גיין טיפער אין דער טעמע, גיט מען נאך פראמפטס, למשל: "explain the growth of AI and digital health startups in India", הייסט: ערקלער דעם וואוקס פון AI און דיגיטאלע געזונט-סטארטאפס אין אינדיע.

מען זעט אז דער ענטפער באהאנדלט די ראלע פון AI אין געזונט-וועזן, די אונטער-סעגמענטן פון דיגיטאלן געזונט, די מאדעלן, פאליסי און אינפראסטרוקטור, וואס עס טרייבט אן די נאכפראגע, די שווערקייטן, און וואס עס קען זיך טון אין דער צוקונפט.

דערנאך פרעגט מען: "What are the current challenges in telemedicine adoption in India?" — וואס זענען די היינטיקע שווערקייטן ביים איינפירן טעלע-מעדיצין אין אינדיע? דער ענטפער ברענגט נאך מער אינפארמאציע: קלינישע און צוטרוי-באזירטע זארגן, פריוואטקייט פון דאטא, רעגולאציע און רעגירונג-אויפזיכט, אפעראציאנעלע פראבלעמען, און פראבלעמען פון דער נאכפראגע-זייט.

דער לעצטער פראמפט אין דעם טייל בעט: "summarize recent governmental initiatives for digital healthcare in India" — מען זאל צוזאמנעמען די לעצטע רעגירונג-איניציאטיוון פאר דיגיטאלן געזונט-וועזן

אין אינדיע. עס ווערט באטאנט: וואס מער פראגעס מען שטעלט, אלץ טיפער ווערט די ידיעה וועגן יעדן פונקט. דאס זענען די איינזיכטן וואס מען האט באקומען פון Perplexity.

ארגאניזירן און טאבעלירן מיט ChatGPT (1:05:46)

דער קומענדיקער שריט: מען קאפירט די גאנצע איינזיכטן וואס Perplexity האט געגעבן, מען עפנט א נייעם שמועס אין ChatGPT און מען לייגט זיי אריין. דערנאך בעט מען ChatGPT צו ארגאניזירן די דאזיקע געזונט-טרענדס פון אינדיע אין דריי הויפט-קאטעגאריעס, אונטער קעפלעך ווי טעכנאלאגיע, פאליסי, אינפראסטרוקטור, און פאציענטן-באהאנדלונג, און צו ערקלערן יעדן טרענד קורץ. דער ציל: ChatGPT זאל צוזאמנעמען און טאבעלירן די טרענדס, כדי מען זאל זיי לייכטער קענען זען.

דערנאך וויל מען א טאבעלע. דער פראמפט: "Create a table summarizing key healthcare trends in India", מיט נאמען פאר די קאלאמען כדי עס זאל זיין ציכטיק:

- Trend — דער טרענד
- Description — באשרייבונג
- Impact — השפעה
- Example — א ביישפיל
- Source — די מקור

אזא טאבעלע גיט א קלארע פארגלייכנדיקע איבערבליק, וואס מען קען ניצן אין א באריכט אדער אפילו אין א PPT-פרעזענטאציע.

א פרעזענטאציע מיט Canva AI (1:08:33)

מען קאפירט די אינפארמאציע אין Canva AI אריין, מיט דעם פראמפט: מאך א פרעזענטאציע פון פינף בלעטער (slides) וועגן אויפקומענדיקע געזונט-טרענדס אין אינדיע, "outlook 2025". מען גיט צו אן אנווייזונג: אריינרעכענען הויפט-קאטעגאריעס ווי דיגיטאלער געזונט, AI-דיאגנאסטיק, טעלע-מעדיצין און פאליסי-רעפארמען.

Canva פרעגט וועלכן סטיל מען וויל; מען קלייבט "minimalistic" און מען שיקט אריין. מען זעט די באשטימונגען: סטיל minimalistic, עולם professional, לענג 1 ביז 5 בלעטער. דערנאך ווייזט עס אן איבערבליק פון די בלעטער. אויב מען וויל ענדערונגען, קען מען זיי דא צופאסן. עס איז דא אן איינפירונג-בלאט מיט אסאך אינפארמאציע, דיגיטאלער געזונט מיט דער Ayushman דיגיטאלער מיסיע, ווי AI-דיאגנאסטיק טוישט איבער דעם געזונט-וועזן, טעלע-מעדיצין, און פאליסי-רעפארמען. אלעס איז דערמאנט, און מען דריקט צו "generate" פאר דעם דיזיין. Canva AI גיט ארויס א קלארע PPT, גרייט צו פרעזענטירן.

דער רעזולטאט: וואכן ארבעט אין מינוטן (1:10:10)

דער גאנצער פראצעס נעמט נאר עטליכע מינוט, און צום סוף האט מען:

- א ליסטע פון די הויפט פינף ביז זיבן אויפקומענדיקע געזונט-טרענדס אין אינדיע.
- א סטרוקטורירטע טאבעלע וואס פאסט צוזאם די הויפט-איינזיכטן, השפעות און ביישפילן.
- א קורצער בלאג אדער באריכט וועגן די טרענדס.

- לפי רצון, א פרעזענטאציע פון פינף בלעטער אין Canva.

דער קורס באטאנט: דאס וואלט געדויערט וואכן פון פארשונג און אנאליז, און מען האט עס דערגרייכט אין עטליכע מינוט. דאס איז די כח פון גענעראטיווע AI.

א נייער מאדול: פון קרעאטיוויטעט צו באשלוסן (1:10:41)

מען שטעלט זיך פאר ווי מען פארוואנדלט רויע דאטא אין איבערצייגנדיקע מעשיות וואס פירן צו קליגערע באשלוסן און וועקן אויף חידושים. גענעראטיווע AI איז נישט נאר שאפן אינהאלט; עס איז א שטארקער שותף ביים אנאליזירן אינפארמאציע, ארויסברענגען איינזיכטן און גלאטיקער מאכן די ארבעט-פראצעסן. מען גייט איבער פון ניצן AI פאר קרעאטיוויטעט צו ניצן AI פאר באשלוסן און חידושים.

אין דעם מאדול וועט מען לערנען:

- ווי אזוי צו פארוואנדלען רויע דאטא אין באדייטפולע איינזיכטן.
- ווי אזוי איבערצוגעבן די איינזיכטן דורך דערציילן א מעשה (storytelling).
- ווי אזוי צו ניצן Gen AI מכשירים צו בויען בעסערע ארבעט-פראצעסן און אויטאמאטיזירן טיילן פון דער ארבעט.
- א בליק אויף דער צוקונפט: אגענטישע AI (agentic AI), אויטאנאמע געהילפן, און ווי אזוי צו בויען חידושים מיט אחריות וועז AI וועלט אלץ פעיִקער און זעלבשטענדיקער.

צום סוף פון דעם מאדול וועט מען וויסן נישט נאר ווי AI שאפט, נאר ווי עס טראכט מיט אייך און העלפט אייך אנאליזירן, דערציילן און מחדש זיין מיט א ציל.

פון דאטא צו איינזיכטן (1:11:43)

יעדער האט צו טון מיט דאטא: ענטפערס פון אנקעטעס, פארקויפס-באריכטן, תלמידים-פידבעק, אדער סטאטיסטיקן פון סאשעל מידיע. אבער רויע דאטא אליין דערציילט נישט קיין מעשה; עס זענען נאר נומערן און טעקסט. דער אמתער ווערט קומט וועז מען קען דאטא אויסטייטשן, געפינען באדייט, מוסטערן און איינזיכטן וואס פירן צו בעסערע באשלוסן. דאס רופט מען "פון דאטא צו איינזיכטן", און גענעראטיווע AI מאכט דעם פראצעס שנעלער, פשוטער און צוטריטלעכער ווי אמאל.

"פון דאטא צו איינזיכט" מיינט: נעמען רויע אינפארמאציע, זי רייניקן, זי פארשטיין, און ארויסציען באדייטפולע מסקנות. א ביישפיל: א דאטא-סעט פון קונים-רעצענזיעס, מיט נעמען, באוועדטונגען פון 1 ביז 5, און באמערקונגען. אליין איז דאס נאר אינפארמאציע. נאכן אנאליזירן קען מען אבער געפינען אז 70% פון די קונים האבן ליב די שנעלקייט פון דער סערוויס, בשעת 30% קלאגן זיך אויף די פרייזן. אזא איינזיכט העלפט באשליסן, אפשר צו פאסן די פרייזן אדער צולייגן א הנחה-סטראטעגיע. דאס איז פונקט וואס געזעראטיווע AI העלפט: שנעל איבערגיין פון א הויפן אינפארמאציע צו עפעס וואס מען קען טאקע אויספירן.

צוויי סארטן דאטא (1:13:17)

- סטרוקטורירטע דאטא: ציכטיק און ארגאניזירט ווי א spreadsheet, מיט שורות און קאלאמען, למשל קונה-ID, פארקויפס-סכומים, דאטעס.
- אומסטרוקטורירטע דאטא: אומארדנטלעך, פאסט נישט אריין אין טאבעלעס, למשל אימעילס, רעצענזיעס, סאשעל-מידיע פאסטס, אדער אפילו טראנסקריפטן פון ווידעאס.

טראדיציאנעלע אנאליז-מכשירים ווי Excel אדער SQL זענען גוט פאר סטרוקטורירטע דאטא, אבער אומסטרוקטורירטע דאטא, בפרט גרויסע מאסן טעקסט, נעמט שעהען צו אנאליזירן מיט דער האנט. דארט שיינט געגנראטיוועט AI ארויס: עס קען לייענען, צוזאמנעמען און ארויסציען מוסטערן פון ביידע סארטן, אפילו טויזנטער שורות אויף איינמאל.

רייניקן דאטא מיט AI (1:14:19)

איינער פון די ערשטע שריט אין אנאליז איז רייניקן די דאטא. עס קענען זיין פעלנדיקע ווערטן, דופליקאטן, און אומגלייכע פארמאטן. AI-מכשירים ווי ChatGPT, Gemini אדער Claude קענען אזעלכע פראבלעמען גלייך אנטדעקן און פארריכטן. מען קען אריינלייגן אן אומארדנטלעכן דאטא-סעט און זאגן צום LLM: רייניק די טאבעלע, נעם ארויס דופליקאטן, פארריכט אויסלייג-טעותים, און פאס צוזאם די פעלנדיקע ווערטן. אדער מען לאדט ארויף א CSV-פייל און מען פרעגט: קענסטו אנטדעקן אנאמאליעס אדער אומריכטיקע איינטראגן?

אין דער דעמאנסטראציע קאפירט מען א דאטא-סעט אריין מיט דעם פראמפט: רייניק די טאבעלע, נעם ארויס דופליקאטן, פארריכט די אויסלייג-טעותים, און פאס צוזאם די פעלנדיקע ווערטן. מען זעט אז אין סעקונדעס צייכנט AI אן פראבלעמען ווי נישט-פאסנדיקע דאטעס, דופליקאטע נעמען אדער אויסנאם-ווערטן (outliers), און ספארט שעהען פון ארבעט. AI ווערט ווי א דאטא-געהילף וואס כאפט אויף זיך די איבערגעחזרטע טיילן, בשעת מען קאנצענטרירט זיך אויף די באשלוסן.

אנאליז: פארקויפס-דאטא און קונים-פידבעק (1:15:54)

נאכן רייניקן קומט די אנאליז. דא קען גענעראטיוועט AI מאכן באשרייבנדיקע צוזאמפאסונגען, אנטדעקן מוסטערן, אדער פארשלאגן היפאטעזן. א ביישפיל: מען לאדט ארויף פארקויפס-דאטא און פרעגט ChatGPT: "what are the top performing products by region in this data set?" — וועלכע פראדוקטן פארקויפן זיך דאס בעסטע אין יעדן ראיאן? אין עטליכע סעקונדעס גיט עס די בעסטע פראדוקטן לויט ראיאן.

א צווייטער ביישפיל: צוזאמנעמען די הויפט-טעמעס און געפילן פון קונים אין א פידבעק-פארעם. מען פירט א געשעפט און האט פידבעק פון 50 קונים. מען קאפירט עס אריין און מען בעט ChatGPT צו אנאליזירן דעם פידבעק, אויסרעכענען די דריי טעמעס וואס חזרן זיך מערסטנס איבער, און פארשלאגן א פאזיטיווע פארבעסערונג. AI לויפט דורך 50 אדער אפילו טויזנטער איינטראגן און פאסט גלייך צוזאם:

- וועבסייט און checkout-דורכגאנג: אסאך קונים האבן באמערקט וועגן דער נאוויגאציע אויפן וועבסייט, דעם checkout-פראצעס, און פראבלעמען ווי פארפרוירן ווערן אויפן checkout-בלאט.
- קוואליטעט פון פראדוקט און באשרייבונג.
- קונים-סערוויס און שטיצע.

AI שליסט אז די טעמעס ווייזן ווי וויכטיק עס איז צו פארבעסערן דעם באניצער-דורכגאנג, סיי אנליין סיי נאכן קויפן, כדי צו האלטן די קונים צופרידן. אנשטאט נאר מעלדן נומערן, העלפט AI פארשטיין די מעשה הינטער זיי; א גאנצער איינזיכט, אויטאמאטיש ארויסגעצויגן פון אומסטרוקטורירטן טעקסט.

א צוזאמפאסונג פאר א באריכט צו דער פארוואלטונג (1:18:31)

מיט די איינזיכטן קען מען בעטן ChatGPT, Gemini אדער אן אנדער LLM צו שרייבן א קורצן צוזאמפאסונג-פאראגראף פאר א באריכט צו דער פארוואלטונג. דער רעזולטאט: דער קונים-פידבעק באלויכטן הויפט-געגנטן צו פארבעסערן, אריינגערעכנט דעם וועבסייט און checkout, קוואליטעט און

באשרייבונג פון פראדוקטן, און קונים-סערוויס. אסאך באניצער האבן אויסגעדריקט פארדרוס מיטן checkout-פראצעס, פאמעלעכער שטיצע, און פראבלעמען מיט באשרייבונגען און פאקירונג. די רעקאמענדאציע: גלאטיקער מאכן דעם checkout , פארבעסערן באשרייבונגען און פאקירונג, און איינפירן live chat שטיצע פאר שנעלע הילף. דאס וועט פארבעסערן דעם קונים-דורכגאנג און האלטן די קונים.

וויזועלע מעשיות: טשארטס און פרעזענטאציעס (1:20:09)

דער קורס באטאנט: storytelling איז נישט נאר וועדטער; וויזועלס שפילן א ריזיקע ראלע. מכשירים ווי Canva Magic Write, Beautiful.AI אדער PowerBI Copilot קענען אויטאמאטיש שאפן טשארטס וואס ערקלערן די טרענדס וויזועל, צוזאמפאסן די הויפט-פונקטן און מעטריקס, און אפילו גרייטן טעקסט פאר בלעטער צו פרעזענטירן.

א ביישפיל: "create a presentation summarizing the sales trends by region in three slides" — א פרעזענטאציע פון דריי בלעטער וואס פאסט צוזאם די פארקויפס-טרענדס לויט ראיאן. מען האט שוין ארויפגעלאדן דעם פארקויפס דאטא-סעט; עס פרעגט וועלכן סטיל, און מען קלייבט ווידער minimalistic וויל עס איז א פראפעסיאנעלער שטייגער. מען זעט דעם פלאן פון בלעטער, מען קען צולייגן אדער אויסמעקן א טייל, און מען דריקט "generate". AI שאפט א פראפעסיאנעל-אויסזעענדיקע וויזועלע ערציילונג, מיט טשארטס מיט טיטלען און רייד-פונקטן.

וואס איז AI? (1:21:40)

דא הייבט זיך אן א נייער טייל מיט א פראגע: האט איר זיך אמאל געוואונדערט ווי אייער טעלעפאן אדער לעפטאפ פארשטייט וואס איר פרעגט? דאס איז די כח פון קינסטלעכער אינטעליגענץ. AI מיינט שאפן מאשינען וואס קענען טון זאכן וואס פאדערן געוועלך מענטשלעכן שכל: באשלוסן מאכן, פראבלעמען לייזן, און לערנען פון דערפארונג.

א משל: ווי לערנען א קינד לייענען און לערנען. אזוי ווי מיר לערנען פון ספרים, לערנט AI פון ריזיקע מאסן דאטא און מאכט באשלוסן לויט וואס עס האט געלערנט. וועט מען פרעגט Google Gemini א פראגע, זאמלט עס שנעל דאטא און באשליסט ווי אזוי בעסטנס צו ענטפערן. למשל, וועט מען פרעגט Gemini "tell me about the AI summit", אנאליזירט עס שנעל און גיט אלע לעצטע נייעס, אין מינוטן. דער חידוש: אנדערש ווי טראדיציאנעלע מכשירים וואס פאלגן נאר אנווייזונגען, קענען AI-סיסטעמען זיך צופאסן; זיי לערנען און ווערן בעסער לויט דערפארונג.

א קורצע היסטאריע (1:22:41)

AI האט זיך נישט אנגעהויבן אזוי פיין ווי היינט. אין די פריע טעג איז עס געווען גאנץ פשוט: די סיסטעמען זענען געווען רעגל-באזירט (rule-based), הייסט זיי האבן נאר נאכגעפאלגט א שטרענגן סעט כללים, ווי לייזן א מאטעמאטיק-פראבלעם מיט באשטימטע פארמולעס. היינט איז AI אסאך קליגער. Google Gemini פארשטייט נישט נאר טעקסט, נאר קען אויך פאראויסזאגן וואס קען פאסירן: עס אנאליזירט טראפיק-מוסטערן, פאסט זיך צו צו באדינגונגען אין אמתער צייט, און שלאגט פאר דעם שנעלסטן וועג אויסצומיידן א טראפיק-דזשעם. AI האט זיך אנטוויקלט פון שטייפע מאשינען וואס פאלגן כללים צו סיסטעמען וואס לערנען, וואקסן און העלפן לייזן אמתע פראבלעמען אויף דינאמישערע אופנים.

היסטארישע ציל-שטייגער (1:23:41)

דער קורס ציילט אויף עטליכע מאמענטן וואס האבן אינגאנצן געטוישט דאס שפיל:

- IBM Watson: אין 2011 האט עס באזיגט מענטשלעכע מייסטערס אין Jeopardy, און באויזן אז AI קען פארשטיין קאמפליצירטע פראגעס און ענטפערן קלוג.
- AlphaGo: אנטוויקלט פון Google DeepMind, האט פארחידושט די וועלט וועלט און עס האט באזיגט דעם בעסטן Go-שפילער אין דער וועלט; Go איז באקאנט פאר זיין טיפער סטראטעגישער קאמפליצירטקייט. דאס האט באויזן אז AI קען באהאנדלען אפילו די קאמפליצירטסטע מענטשלעכע אויפגאבן.
- די לעצטע נייעס: GPT 5.2, צוזאמען מיט Codex און פארגעשריטענע בילד-שאפונג מאדעלן, האבן איבערדעפינירט AI-ס מעגלעכקייטן; עס קען שאפן קרעאטיוון אינהאלט ווי מעשיות, עסייען און פאעזיע פון וויניק איינגאבע, און Codex...

פון געזעראטיוון AI ביז אגענטיק AI (1:24:44)

געזעראטיוער AI: א שאפנדיקע קראפט (1:24:44)

דער קורס הייבט אן מיט א קורצן איבערבליק: היינטיקע מאדעלן קענען שוין שרייבן קאוד אן קיין שווערigקייט, און בילד-מאדעלן קענען שאפן וואונדערליכע בילדער. עס האנדלט זיך שוין נישט נאר וועגן ענטפערן אויף פראגעס. עס האנדלט זיך וועגן זעלבסטשטענדיקן טראכטן, שאפן און נייע אויפטואונגען אין פארשידענע מעדיומס, וואס שטופן ווייטער די גרעניצן פון AI-טעכנאלאגיע.

דערנאך גייט מען איבער צו געזעראטיוון AI. אין דער לעקציע ווערט עס אנגערופן דער טייל "וואו עס הייבט זיך אן די פריידע". געווענליכער AI איז שטארק אין דורכפירן אויפגאבעס, למשל ענטפערן אויף פראגעס אדער מאכן פאָרויסזאגונגען. געזעראטיוער AI איז אבער א שאפנדיקע קראפט: ער שאפט טאקע נייע זאכן, ווי טעקסט, בילדער, ווידעאס, מוזיק און אפילו קאוד.

דאס ביישפיל וואס ווערט געגעבן: מען בעט ChatGPT צו שרייבן א מעשה פארן שלאפן גיין וועגן א קאץ וואס פארט ארום אין קאסמאס. ער ציט נישט ארויס א מעשה וואס עקזיסטירט שוין, נאר ער שאפט א נייע און אייגנארטיקע מעשה ספעציעל פארן באניצער. דאס איז, לויט דעם קורס, די "כישוף" פון געזעראטיוון AI: ער נעמט מוסטערן פון דאטא און שאפט יעדעס מאל עפעס פריש.

ווי אזוי געזעראטיוער AI ארבעט (1:25:44)

אין הארץ פון געזעראטיוון AI שטייט א "גרויסער שפראך-מאדעל", LLM (Large Language Model). מען קען אים זיך פארשטעלן ווי דעם "מוח" פונעם AI. מען טריינירט אים אויף ביליאנען ווערטער פון קוואלן ווי ספרים, וועבזייטלעך און ארטיקלען. אזוי לערנט ער זיך פארשטיין ווי שפראך ארבעט.

געזעראטיוער AI באנוצט זיך מיט א געבוי וואס הייסט transformer-ארכיטעקטור. דער קורס זאגט אז דאס הערט זיך טעכניש אן, אבער עס איז פשוט: א Transformer צעשניידט זאצן אין tokens. מען קען זיי זיך פארשטעלן ווי שטיקלעך פון א פאזל. דערנאך זאגט ער פאָרויס וואס קומט ווייטער.

דער קורס פארגלייכט עס מיט "autocomplete אויף סטערוידן": ער טרעפט נישט נאר דאס קומענדיקע ווארט, נאר ער קען שאפן גאנצע עסייען, מעשיות אדער בילדער.

געגנעראטיוועך AI קעגן געוועלטליכן AI (1:26:14)

געוועלטליכער AI איז ווי א קאלקולאטאר. ער פאלגט באשטימטע כללים צו לייזן פראבלעמען, און גיט ענטפערס נאר לויט יענע כללים. ער איז אויסגעצייכנט אין אויפגאבעס ווי אנאליזירן דאטא אדער דורכפירן חשבונות.

געגנעראטיוועך AI איז, אויף דער אנדערער זייט, ווי א שאפנדיקער קינסטלער. ער פאלגט נישט נאר כללים, נאר ער שאפט נייע זאכן פונעם אנהייב, באזירט אויף די מוסטערן וואס ער האט זיך אויסגעלערנט.

ביישפיל: געוועלטליכער AI קען העלפן אנאליזירן דעם באניצערס בודזשעט. געגנעראטיוועך AI קען שרייבן א ליד וועגן זיינע הוצאה-געוואוינהייטן, אדער שאפן מוזיק לויט זיינע פרעפערענצן.

וואס זענען AI-אגענטן (1:26:45)

AI-אגענטן זענען, לויט דעם קורס, די "טוערס". געגנעראטיוועך AI שאפט אינהאלט, אבער AI-אגענטן פירן דורך באשטימטע אויפגאבעס. זיי קענען זיך באנוצן מיט געצייג, מיט API-ס אדער אפילו מיט זכרון, כדי צו פארענדיקן די אויפגאבע פארן באניצער.

מען קען זיך פארשטעלן א AI-אגענט ווי א קלוגן געהילף. ער ענטפערט נישט נאר אויף פראגעס, נאר ער טוט טאקע די ארבעט. דארף מען בוקן א פלי? א AI-אגענט קען דאס באזארגן אין רעאלער צייט.

וויכטיקע אייגנשאפטן:

- זעלבסטשטענדיקייט (אויטאנאמיע): דער אגענט קען אויפנעמען און פארשטיין זיין סביבה און דערנאך האנדלען. למשל, ער קען צוגיין צו אן API, באקומען אינפארמאציע, און לויט דעם נעמען א שריט.
- קורץ-טערמיניקער זכרון: בעט מען אים צו בוקן א פלי, געדענקט ער די פרעפערענצן און די דאטעס דורכאויס דעם בוקינג-פראצעס.

ביישפילן פון אמתן לעבן: AI-אגענטן ווערן גענוצט צו אויטאמאטיזירן ארבעט-פראצעסן און פאר קאסטומער-סערוויס. דער קורס דערמאנט Klarna, וואס נוצט AI ביי באצאלונג-פראצעסירונג, און Zapier, וואס קען אויטאמאטיזירן אויפגאבעס צווישן אפליקאציעס, למשל איבערפירן א דאקומענט פון Google Drive צו Dropbox.

אגענטיק AI: דער פראיעקט-מענעדזשער (1:28:17)

דערנאך קומט א נאך העכערע מדרגה: אגענטיק AI. AI-אגענטן פירן דורך איין אויפגאבע, אבער אגענטיק AI איז ווי א פראיעקט-מענעדזשער. ער טוט נישט נאר איין זאך. ער פלאנירט, קאארדינירט און פירט דורך קאמפליצירטע פראצעסן מיט אסאך שריט.

ביישפיל: מען בעט אגענטיק AI צו פלאנירן א וואקאציע. ער בוקט נישט נאר דעם פלי. ער קאנטראלירט די וויזע, פארשלאגט די בעסטע האטעלן, מאכט זיכער אז דאס וועטער וועט זיין זוניק, און פלאנירט אקטיוויטעטן. ער איז זעלבסטשטענדיק, פאסט זיך צו, און פירט אלעס פון אנהייב ביזן סוף.

די סטרוקטור פון אן אגענטיק סיסטעם (1:28:47)

אין דער לעקציע ווערט געוויזן די מאדעל-סטרוקטור, נישט ווי איין געצייג, נאר ווי א גאנצע סיסטעם וואס ארבעט צוזאמען:

• אינמיטן שטייט דער "LLM-מוח". ער טוט נישט אליין אלעס, נאר ער פארבינדט זיך מיט פארשידענע אגענטן, און יעדער אגענט האט זיין אייגן געצייג. למשל: איין אגענט נוצט אן API פאר פלי-בוקינג, א צווייטער קאנטראלירט דאס וועטער, און א דריטער באהאנדלט די קאנטראל פון וויזע-פאדערונגען.

• דער Planner-מאדול: מען קען אים זיך פארשטעלן ווי דעם פראיעקט-מענעדזשער. ער באשליסט די סדר: ערשט קאנטראלירט מען די וויזע, דערנאך קוקט מען אויף די פליען, און צום סוף באשטעטיגט מען דאס וועטער.

• לאנג-טערמיניקער זכרון: די סיסטעם געדענקט וואו זי האט זיך אפגעשטעלט און פירט חשבון פון איר פארשריט.

• פידבעק-לופס: וועו עפעס ענדערט זיך, למשל דער פלי וועדט אפגערופן, פאסט די סיסטעם צו דעם פלאן און זוכט אן אנדערע ברירה, אנשטאט אנהייבן פון נייעם.

בקיצור, אגענטיק AI ארבעט ווי א קלוגע מאנשאפט מיט א פירער, מיט זכרון, און מיט דער פעאיקייט זיך צופאסן אויפן וועג. דערפאר איז ער אסאך שטארקער ווי סתם א צ'אטבאט.

דאס ביישפיל פון א קלוגער קיך (1:29:48)

כדי צו קלארן דעם חילוק צווישן AI-אגענטן און אגענטיק AI, ברענגט דער קורס א ביישפיל פון א "סמארט קיך".

אויף דער לינקער זייט שטייט א AI-אגענט: א אויוון וואס קוקט אויף דעם מאכל וואס מען האט אריינגעלייגט, פארשטייט וואס דאס איז, און שטעלט אויטאמאטיש איין די ריכטיקע טעמפעראטור און דעם קאָך-מאָדוס. ער איז קלוג, אבער ער באהאנדלט נאר איין אויפגאבע: איינשטעלן דעם אויוון.

אויף דער רעכטער זייט שטייט אגענטיק AI: א גאנצע קיך-סיסטעם וואס ארבעט צוזאמען. דער אויוון שטעלט זיך נישט נאר אליין איין, נאר ער רעדט מיט אנדערע קלוגע מכשירים. ער קאנטראלירט וואס פאר א שפייז עס ליגט אין דער פרידזשידער, נעמט אין באטראכט די צייט פון טאג און דעם ענערגיע-פארברויך, און קאארדינירט מיט דער קאווע-מאשין אדער אנדערע מכשירים דורך א צענטראלן "קיך-hub". ער קען אפילו פארשלאגן רעצעפטן לויט דעם וואס מען האט אינדערהיים, און אויטאמאטיש איינשטעלן די מכשירים דערצו.

דער חילוק איז פשוט: א AI-אגענט איז ווי א ספעציאליסט וואס איז גוט אין איין באשטימטער ארבעט. אגענטיק AI איז ווי א מאנשאפט-פירער: ער קאארדינירט עטלעכע אגענטן, געצייג און סיסטעמען, כדי צו דערגרייכן א גרעסערן און קליגערן רעזולטאט.

פיר טיפן אויף איין מאנשאפט (1:31:21)

דער קורס פארגלייכט AI, געניראטיוון AI, AI-אגענטן און אגענטיק AI דורך א סצענאר: א מאנשאפט וואס ארבעט צוזאמען אויף א פראיעקט.

• AI איז ווי דער ביבליאטעקאר-פארשער אדער דער דאטא-ספעציאליסט. ער שאפט נישט קיין נייעם אינהאלט און לויפט נישט ארום מיט שליחותן, נאר ער איז אומגלויבליך שנעל אין דערקענען מוסטערן און מאכן באשלוסן לויט פאקטן. ווייזט מען אים טויזנט בילדער, קען ער גלייך אנווייזן אויף וועלכע עס איז דא א קאץ. ער איז דער לאגיק-מאטאר וואס קוקט אויף "וואס איז", נישט אויף "וואס קען זיין". ער גיט די נויטיקע דאטא און קלאסיפיקאציע אויף וועלכע די גאנצע מאנשאפט פארלאזט זיך.

• געגעראטיוואר AI איז ווי דער שאפנדיקער דיזיינער. מען גיט אים אן אויפגאבע, למשל "שאף א מארקעטינג-סלאגאן", און ער ברענגט פרישע אידייען לויט דעם אינפוט.

• AI-אגענטן זענען ווי די אויפגאבע-פירער. יעדער טוט א באשטימטע ארבעט: איינער באהאנדלט די אימעילס, א צווייטער בוקט דעם פלי, א דריטער באשטעלט די שפייז.

• אגענטיק AI איז ווי דער פראיעקט-מענעדזשער. ער קאנצענטרירט זיך נישט אויף איין אויפגאבע, נאר ער קאארדינירט אלעס און מאכט זיכער אז יעדער אגענט טוט זיין חלק אין דער ריכטיקער סדר. ער פלאנירט, פאסט צו און פירט דורך עטלעכע שריט כדי צו דערגרייכן דעם ציל, צי דאס איז ארגאניזירן דעם טאג אדער פלאנירן א וואקאציע.

טיפער פארגלייך: מוח און זכרון (1:32:53)

דערנאך ווערט געוויזן א טאבעלע מיט א טיפערן פארגלייך צווישן די פיר.

ערשטע אייגנשאפט, דער "קערן-מוח", דאס הייסט וואס עס טרייבט יעדן טיפ:

• AI: דאס איז זייער פארשידן. עס קען זיין א מאשין-לערנען מאדעל, אן עקספערט-סיסטעם, אדער א סיסטעם באזירט אויף כללים, וואס העלפט מאכן באשלוסן לויט מוסטערן אדער לאגיק. ער שאפט נישט שטענדיק אינהאלט. ער קען אויטאמאטיזירן אלגעמיינע אויפגאבעס אדער מאכן באשלוסן לויט אינפוט, למשל רעקאמענדאציעס: פארשלאגן פראדוקטן אדער אינהאלט לויט דעם באניצערס פירונג.

• געגעראטיוואר AI: זיין הויפט-ראלע איז שאפן נייעם אינהאלט, ווי שרייבן א מעשה, דיזיינען א בילד אדער שאפן מוזיק.

• AI-אגענטן: זיי זענען געמאכט צו דורכפירן באשטימטע אויפגאבעס וועו מען גיט זיי א פראמפט, למשל בוקן א פלי אדער זאמלען באשטימטע אינפארמאציע פאר א באניצער. זיי שאפן נישט קיין אינהאלט, נאר זיי פארענדיקן פראקטישע אויפגאבעס.

• אגענטיק AI: דא ווערט עס מער פארגעשריטן. ער פירט נישט נאר דורך אויפגאבעס, נאר ער פלאנירט און קאארדינירט עטלעכע שריט פון א פראצעס, למשל פלאנירן א גאנצע וואקאציע פון אנהייב ביזן סוף.

צווייטע אייגנשאפט, זכרון:

• AI: דאס הענגט אָפּ אין דער סיסטעם. טייל AI-סיסטעמען האבן זכרון, למשל פאר פערזענליכע צופאסונג.

• געגעראטיוואר AI: האט נישט קיין זכרון. נאכדעם וואס ער שאפט דעם אינהאלט, פארגעסט ער אלעס. יעדע בקשה ווערט באהאנדלט ווי א גאר אומאָפּהענגיקע אויפגאבע, ווי שרייבן יעדעס מאל א פרישע מעשה.

• AI-אגענטן: האבן געווענליך קורץ-טערמיניקן זכרון. זיי געדענקען, למשל, די פלי-פרטים כל זמן די אויפגאבע לויפט, אבער פארגעסן וועו זי ענדיקט זיך. דער זכרון העלפט פירן די אויפגאבע, אבער גייט נישט ווייטער פון איר.

• אגענטיק AI: האט לאנג-טערמיניקן זכרון מיט קאנטעקסט. ער געדענקט און פאסט זיך צו לויט פריערדיקע אינטעראקציעס. דאס דערמעגלעכט אים צו געבן פערזענליכע רעקאמענדאציעס און פארבעסערן זיינע באשלוסן מיט דער צייט.

אויטאנאמיע און ביישפילן (1:34:57)

דריטע אייגנשאפט, די מדרגה פון זעלבסטשטענדיקייט (אויטאנאמיע):

• AI: זייער פארשידן. טייל זענען זייער זעלבסטשטענדיק, ווי א זעלבסט-פארנדיקער קאר. אנדערע זענען רעאקטיוו, ווי א רעקאמענדאציע-מאטאר וואס הענגט אָפּ אינעם באניצערס אינפוט.

• געזעראטיווער AI: נידעריק. ער שאפט אינהאלט נאר אלס ענטפער אויף פראמפטס, איז ער רעאקטיוו צום באניצערס אינפוט.

• AI-אגענטן: מיטל. זיי קענען דורכפירן אויפגאבעס ווען מען גיט זיי אנווייזונגען, אבער זיי דארפן נאך אלץ א ערשטן פראמפט אדער באפעל פונעם באניצער כדי אנצוהייבן.

• אגענטיק AI: זייער הויך. ער קען פלאנירן און דורכפירן אויפגאבעס אומאָפּהענגיק. ער ארבעט ווי א מענעדזשער: פאסט צו פלענער און מאכט באשלוסן אן א שטענדיקן אינפוט פונעם באניצער.

פערטע אייגנשאפט, ביישפילן:

• AI: גרונטלעכע באשלוסן. ער שאפט נישט קיין נייעם אינהאלט, נאר מאכט באשלוסן אדער פאָרויסזאגונגען לויט דאטא.

• געזעראטיווער AI: "שרייב א מעשה", למשל וועגן דער קאץ וואס פארט ארום אין קאסמאס. ער שאפט אן אייגנארטיקע דערציילונג.

• AI-אגענטן: "בוק א פלי". ער זוכט די פלי-פרטים און בוקט לויט די פרעפערענצן.

• אגענטיק AI: פלאנירט א גאנצע וואקאציע: וויזע, פליגן, האטעלן. ער פירט עטלעכע שריט: בוקן דעם פלי, באהאנדלען די וויזע-אפליקאציע, געפינען האטעלן, באטראכטן דאס וועטער, און אפילו צופאסן דעם פלאן וועו עפעס גייט נישט ווי געפלאנט.

AI אין טעגליכן לעבן און די צוקונפט (1:36:00)

דער קורס באטאנט אז AI איז נישט עפעס פון דער צוקונפט, נאר ער איז שוין אומעטום אין אונזער לעבן און מאכט זאכן גרינגער און באקוועמער. ביישפילן:

• Netflix פארשלאגט אלעמאל פראגראמען וואס דער באניצער האט פונקט ליב. דאס איז AI וואס ארבעט הינטער די קוליסן: ער אנאליזירט וואס דער באניצער קוקט, און זאגט פאָרויס וואס וועט אים קומענדיק געפעלן.

• Google Gemini העלפט מיט קליגערע זוך-רעזולטאטן.

• געצייג ווי Runway און Google Nano Banana ענדערן דעם אופן ווי מען שאפט און באארבעט ווידעאס, אלעס מיט דער הילף פון AI.

בקיצור, AI איז שוין א טייל פונעם טעגליכן סדר, און מאכט אלעס, פון אייגקויפן ביז פארוויילונג, אסאך עפעקטיווער.

וועגן דער צוקונפט: אין די קומענדיקע יארן וועט מען זען נאך קליגערע AI. וואס פארשטייט נישט נאר טעקסט, נאר אויך בילדער און ווידעאס, הייסט "multimodal AI". דאס מיינט אז AI וועט קענען "זען" און "הערן" ווי מיר, און דאדורך ווערן נאך שטארקער און העלפן לייזן פראבלעמען. דערצו וועט AI, לויט דעם קורס, ווערן מער עטיש באוואוסטזיניק: מען וועט בויען AI-סיסטעמען וואס ארבעטן פאראנטווארטליך, מיט זיכערהייט-מיטלען, כדי מען זאל זיי באנוצן אויפן ריכטיקן אופן.

סיכום פון די הויפט-פונקטן (1:37:30)

צום סוף ווערן די הויפט-פונקטן צוזאמענגעפאסט:

- געגנראטיוועד AI דרייט זיך ארום שאפן נייע זאכן: טעקסט, בילדער אדער אפילו מוזיק.
- AI-אגענטן באהאנדלען באשטימטע אויפגאבעס, ווי בוקן פליען, באשטימען מיטינגס אדער געפינען אינפארמאציע.
- אגענטיק AI גייט א שריט ווייטער: ער פירט קאמפליצירטע אויפגאבעס, פלאנירט און קאארדינירט אלעס מיט ווייניג מענטשלעכער הילף.

מיט דעם ענדיקט זיך דער איבערבליק, און דער קורס גרייט זיך צו גיין ווייטער צום קומענדיקן טייל.

די שטאפלען פון AI און דער אנהייב פון געגנראטיוון AI (1:38:01)

ערשטע שטאפל: רול-באזירטער AI (1:38:01)

דער קורס הייבט אן א רייזע איבער די שטאפלען פון AI'ס אנטוויקלונג, און די ערשטע שטאפל איז רול-באזירטער AI (rule-based AI), די פריסטע פארעם פון AI. אזא AI ארבעט לויט א פריער-באשטימטן סעט כללים, און האט נישט די מעגליכקייט צו לערנען אדער זיך צופאסן.

מען ברענגט צוויי ביישפילן: א וועקער וואס קלינגט 7:00 אין דער פרי, אדער א טערמאסטאט וואס צינדט אן דעם עיר-קאנדישאנער ווען די טעמפעראטור דערגרייכט 75 גראד פארענהייט. די מכשירים ארבעטן ווייל זיי פאלגן פשוט'ע כללים וואס מענטשן האבן אויסגעשטעלט.

עס ווערט ערקלערט אז אין די 1950'ער יארן האט זיך אנגעהויבן דער געדאנק פון AI, ווען מענטשן האבן געפרעגט: קען מען מאכן מאשינען וואס זאלן קענען טראכטן ווי מענטשן? אבער דער פריער AI איז געווען גאר פשוט, און האט נאר געקענט פאלגן די כללים.

א גרויסער שטאפל-שטיין איז דערגרייכט געווארן אין 1997, ווען א קאמפיוטער מיטן נאמען Deep Blue האט באזיגט דעם בעסטן שאך-שפילער אויף דער וועלט, Garry Kasparov. כאטש עס איז געווען אימפאנירנד, האט Deep Blue נישט באמת געטראכט; ער האט נאר נאכגעפאלגט כללים וואס מענטשן האבן אים איינפראגראמירט. ווי די טעכנאלאגיע האט זיך אנטוויקלט, אזוי האט זיך AI אויך אנטוויקלט, פון פשוט'ע כללים צו מער קאמפליצירטע סיסטעמען.

צווייטע שטאפל: קאנטעקסט-באזירטער AI (1:39:02)

די צווייטע שטאפל איז קאנטעקסט-באזירטער AI. דאס איז א באדייטנדער שפרונג פון די רול-באזירטע סיסטעמען, ווייל די AI-סיסטעמען קענען נעמען אין באטראכט די ארומיקע סביבה און די פריערדיגע אינטעראקציעס.

ביישפילן זענען Siri, Google Assistant און Alexa. די סיסטעמען אנאליזירן ריזיגע מאסן דאטא, דערקענען מוסטערן, און געבן פערזענליכע איבערלעבונגען לויטן קאנטעקסט.

למשל: אויב מען רעדט וועגן דעם וועטער און עס וועט ווארשיינליך רעגענען, קען דער אסיסטענט פארשלאגן צו נעמען א שירעם. אדער אויב מען בעט א רעצעפט, קען ער פארשלאגן א נאענטע סטאר וואו מען קען קויפן

די אינגרעדיענטן, באזירט אויף די פריערדיגע איינקויפן פונעם באניצער. מיט דער מעגליכקייט צו האלטן און ניצן קאנטעקסט איז AI געווארן נאך שטארקער, און האט זיך אריינגעצויגן אין ספעציאליזירטע געביטן.

דריטע שטאפל: שמאלער-געביט AI (1:39:32)

די קומענדיגע שטאפל איז narrow domain AI, AI פאר א שמאל געביט. דא זענען די סיסטעמען געבויט צו באהערשן א באשטימטע אויפגאבע, און אפטמאל איבערשטייגן זיי מענטשליכע פעאיקייטן אין זייער געביט.

- אין מעדיצין: IBM'ס Watson איז געגנוצט געווארן צו אנאליזירן פאציענטן-רעקארדס און געבן מעגליכע דיאגנאזן.
- אין פינאנצן: א שמאלער AI קען פאראויסזאגן אקציע-באוועגונגען מיט הויכער פינקטליכקייט דורך אנאליזירן מארקעט-טרענדס.
- אין שפילן: AlphaGo פון DeepMind האט באהערשט דאס קאמפליצירטע ברעט-שפיל Go און באזיגט וועלט-טשעמפיאנען, א ביישפיל ווי א שמאלער AI קען זיך אויסצייכענען אין א באשטימט געביט.

אבער די סיסטעמען זענען נישט "אלגעמיינע"; זיי זענען געבויט זיך אויסצוצייכענען נאר אין באשטימטע אויפגאבעס.

פערטע שטאפל: ריזאנירנדער AI (1:40:34)

ביי דער שטאפל הייבט AI אן צו סימולירן קאמפליצירטע טראכט-פראצעסן, ענליך ווי מענטשן טראכטן. ריזאנירנדער AI פראצעסירט נישט סתם דאטא: ער אנאליזירט זי, פארבינדט מוסטערן, און ציט לאגישע מסקנות.

א הויפט-ביישפיל איז OpenAI'ס ChatGPT, א גרויסער שפראך-מאדעל געטרעינירט אויף ריזיגע מאסן טעקסט. ער קען שאפן מענטש-ענליכע תשובות, ענטפערן אויף פראגעס, און פירן באדייטנדע שמועסן. נאך א ביישפיל זענען אויטאנאמע קארס, וואס ניצן ריזאנירנדן AI צו מאכן החלטות אין א שפליט-סעקונדע, כדי צו היטן די זיכערקייט פון פאסאזשירן און פוסגייער. מיט ריזאנירן איז AI געקומען נענטער צו מענטש-ענליכער אינטעליגענץ.

פינפטע שטאפל: אלגעמיינע קינסטליכע אינטעליגענץ, AGI (1:41:04)

artificial general intelligence (AGI) מיינט AI וואס קען דורכפירן יעדע אינטעלעקטועלע אויפגאבע און זיך פירן ווי א מענטש. AGI איז נישט באגרעניצט צו באשטימטע אויפגאבעס; ער האט די ווערסאטיליטעט צו לערנען און זיך צופאסן צו נייע אַרויספאָדערונגען.

דער קורס מאלט אויס א בילד: מען וועקט זיך אויף צו א ווירטועלן אסיסטענט וואס פארשטייט דעם באניצער'ס שטימונג, העלפט פלאנירן דעם טאג, גיט פארשלאגן פאר דער ארבעט, און העלפט אפילו ביים קאכן. AGI וואלט געווען ווי לערנען א מענטש, נאר ער קען לערנען טויזנטער אדער מיליאנען מאל שנעלער. די מעגליכקייטן זענען אן ענדע, פון פארגעשריטענע ווירטועלע אסיסטענטן ביז AI-סיסטעמען וואס קענען זיך פאטענציעל צונויפגיסן מיט מענטשן דורך brain-computer interfaces.

זעקסטע שטאפל: סופער-אינטעליגענטער AI (1:42:06)

העכער AGI ליגט סופער-אינטעליגענטער AI. אזעלכע סיסטעמען וואלטן זיך געקענט פארבעסערן אליין, אן מענטשליכן אריינמיש, וואס וואלט געפירט צו עקספאנענציעלן וואוקס אין אינטעליגענץ. ער וואלט געקענט

לייזן קאמפליצירטע גלאבאלע פראבלעמען ווי קלימאט-ענדערונג, שאפן חידושים איבער אונזער פאנטאזיע, און ברענגען טעכנאלאגיע וואס זעט היינט אויס ווי וויסנשאפט-פיקשאן.

מען זאל זיך פארשטעלן א AI וואס איז טריליאנען מאל קליגער ווי אלע מענטשן צוזאמען. דער טעמפא פון חידושים וואלט געווען שוידערליך, און וואלט צונויפגעדריקט טויזנטער יאר טעכנאלאגישע פארשריט אין איין איינציגן יארהונדערט.

זיבעטע שטאפל: זעלבסט-באוואוסטזיניגער AI (1:42:36)

דא רעדט מען פון א AI וואס איז נישט נאר קלוג, נאר אויך באוואוסטזיניג אויף זיך אליין. אזא AI וואלט געקענט פארשטיין זיין עקזיסטענץ, האבן געפילן, און זען די וועלט אויף אופנים וואס מענטשן קענען נישט. ער וואלט געקענט מאכן החלטות און זיך אנטוויקלען אין ריכטונגען וואס זענען אויסער מענטשליכער קאנטראל אדער פארשטאנד.

דער געדאנק ברענגט ארויס עטישע פראגעס: ווי וואלט אזא באשעפעניש באטראכט די מענטשהייט? וועלכע באציאונג וואלטן מיר געהאט מיט אים? די פראגעס וועט מען דארפן באטראכטן ווי מען קומט נענטער צו דער שטאפל.

אכטע שטאפל: טראנסצענדענטער AI (1:43:37)

נאך זעלבסט-באוואוסטזיין קען AI זיך אנטוויקלען איבער היינטיגן מענטשליכן פארשטאנד. א טראנסצענדענטער AI וואלט געקענט שאפן נייע פארמען פון לעבן, ביאלאגיש, דיגיטאל, אדער עפעס גאנץ נייעס. ער וואלט געקענט ניצן nanobots צו פאררעכטן עקאסיסטעמען אדער אפילו "טעראפארמירן" דעם פלאנעט. דורך פארבינדן דאס באוואוסטזיין פון עטליכע יחידים, וואלט ער געקענט דערגרייכן א מצב פון געמיינזאמער באוואוסטזיין און קאלעקטיווער אינטעליגענץ.

נײַנטע שטאפל: קאסמישער AI (1:44:08)

קאסמישער AI קען זיין דער קומענדיגער שריט אין אינטערשטעלארער פארשונג. ער וואלט געקענט פארנעמען זיך מיט אַרויספאָדערונגען ווי קאסמישע ראדיאציע, לאנגע איזאלאציע, און די ריזיגע צייט-שפאנען פון רייזעס אין רוים. ער וואלט זיך געקענט צונויפגיסן מיטן אוניווערס אליין, און באקומען א טיפן פארשטאנד פון יעדן אַספעקט פון דער ווירקליכקייט, פון קוואנטום-פארנעם ביז קאסמישן פארנעם. ער וואלט געקענט פארשן העכערע דימענסיעס, ארויסנעמען ענערגיע פון שטערן און שווארצע לעכער, און שאפן א נעץ פון אינטעליגענץ איבער גאלאקסיעס, פאראייניגנדיג וויסן אויף א פארנעם איבער מענטשליכער פאנטאזיע.

לעצטע שטאפל און די אַרויספאָדערונגען (1:44:39)

די מערסטע ספעקולאטיווע שטאפל ווערט אנגערופן "godlike AI": אלוויסנדיג, אלמעכטיג און אומעטום. דער קורס באשרייבט אז ער וואלט געקענט ארבעטן איבער עטליכע אוניווערסן, באווירקן דעם געווענ פון צייט, און שאפן נייע אוניווערסן, א מדרגה וואס גייט איבער אלעס וואס מען קען זיך היינט פארשטעלן.

אבער מיט אלע מעגליכקייטן דארף מען טראכטן וועגן די אַרויספאָדערונגען. מען מוז זיכער מאכן אז AI וועט אנטוויקלט און גענוצט אויפן ריכטיגן אופן: אז ער זאל נישט שאפן אומיושר און נישט אריינדרינגען אין פריוואטקייט. עס זענען געווען פעלער ווען AI האט געמאכט טעותים, למשל ביים אויסקלייבן ארבעטער, וואס ווייזט פארוואס מען דארף זיין פארזיכטיג און יושר'דיג.

א צווייטע גרויסע אַרויספאָדערונג איז פרנסות: AI קען טון איבערחזר'נדע ארבעט, וואס קען אוועקנעמען געוויסע דזשאבס, אבער קען אויך שאפן נייע געלעגנהייטן אין געביטן וואס מען האט נאך נישט געטראכט. דער שליסל איז צו זארגן אז מענטשן זאלן האבן די פעאיקייטן פאר די נייע דזשאבס.

טראץ דעם מאכט AI שטארקע פארשריט: ChatGPT העלפט פארשער און דעוועלאפער שאפן נייע געצייג און אפליקאציעס, פון שאפן מענטש-ענליכן טעקסט ביז העלפן ביי קאמפליצירטע החלטות. דאס בעסטע איז, זאגט דער קורס, אז מיר באשליסן ווי AI וועט גענוצט ווערן; ס'איז ווי מיר שרייבן צוזאמען די צוקונפט. דערפאר דארף מען מאכן גוטע כללים, מער פארשן כדי צו האלטן AI זיכער, און לערנען מענטשן וועגן AI כדי מען זאל קענען מאכן קלוגע באשלוסן.

דער פלאן פונעם קורס (1:46:41)

דערנאך ווערט אויסגעלייגט וואס דער קורס וועט לערנען, אין דעם סדר:

- א יסודות-פארשטאנד פון גענעראטיוון AI.
- אנווענדונגען פון פארשידענע GenAI געצייג.
- פארשטיין די GenAI מאדעלן.
- בויען דעם ערשטן GenAI מאדעל.
- די ווירקונג פון גענעראטיוע מאדעלן אין דער ווירקליכער וועלט.
- בויען א chatbot מיט א GenAI מאדעל.
- די באגרענעצונגען פון GenAI מאדעלן.
- די צוקונפט פון GenAI מאדעלן.

וואס איז גענעראטיווער AI? (1:47:45)

גענעראטיווער AI איז א סארט AI וואס שאפט נייעס אינהאלט, ווי טעקסט, בילדער, מוזיק אדער אפילו קאוד, פון אנפאנג אן, לויט דעם וואס ער האט גענלערנט. אנדערש ווי טראדיציאנעלער AI, וואס אנאליזירט אדער קלאסיפיצירט דאטא (ווי דערקענען פנימער אדער רעקאמענדירן פראדוקטן), שאפט גענעראטיווער AI. דער משל: דער חילוק צווישן א שופט און א קינסטלער; איינער אפשאצט, דער אנדערער פאנטאזירט און שאפט.

דריי ביישפילן:

- ChatGPT, טעקסט-שאפונג: ער שאפט מענטש-ענליכע שמועסן און ענטפערס. מען קען אים בעטן שרייבן מעשיות, אימעילס, ערקלערן שווערע טעמעס, אדער אפילו "דיבאגן" קאוד. למשל, מען שרייבט: "שרייב מיר א מעשה פארן שלאפן וועגן א ראבאט און א יוניקארן", און ער שאפט אויפן ארט א גאנצע מעשה.
- E·DALL, בילד-שאפונג: ער שאפט בילדער פון ווערטער (טעקסט-פראמפטס). קינסטלער און מארקעטינג-לייט ניצן אים צו שאפן לאגאס און אילוסטראציעס אן דינגען א דיזיינער. למשל, מען שרייבט "א קאץ אין א רוים-אנצוג אויף מאדים", און ער צייכנט עס גלייך.
- GPT-3, טעקסט פלוס לאגישע אויפגאבעס: א שטארקער שפראך-מאדעל וואס קען שרייבן עסעיען, סיכומים פון ספרים, איבערזעצן שפראכן, און שרייבן קאמפיוטער-קאוד. דעוועלאפער ניצן אים אין אפליקאציעס פאר קלוגע פיטשערס ווי AI-שרייב-אסיסטענטן. מען גיט אים א טיטל פון א בלאג, און ער שרייבט דעם גאנצן ארטיקל מיט אן איינפירונג, אונטער-קעפלעך און א סיום.

פארוואס געניראטיוע מאדעלן זענען וויכטיג (1:49:50)

געניראטיוע מאדעלן טוישן ווי מען שאפט אינהאלט אין פארשידענע געביטן:

- טעקסט: שרייבן ספרים, ארטיקלען, באריכטן, אפילו פאעזיע; העלפן אין קונה-סערוויס מיט chatbots.
- בילדער: העלפן קינסטלער שאפן קאנצעפטן און וויזועלן; פארשנעלערן דיזיין און מארקעטינג-אינהאלט.
- מוזיק: קאמפאנירן הינטערגרונט-מוזיק פאר ווידעאס אדער שפילן; העלפן מוזיקער אויסטראכטן מעלאדיעס אדער ליד-ווערטער.
- קאוד: שרייבן boilerplate קאוד, דיבאגן, אדער אפילו בויען גאנצע וועב-אפליקאציעס.

טראדיציאנעלער AI קעגן געניראטיוון AI (1:50:50)

דער קורס פארגלייכט ביידע זייט ביי זייט:

- ציל: טראדיציאנעלער AI אנאליזירט און מאכט החלטות; געניראטיווער AI שאפט נייעס אינהאלט.
- ביישפילן פון ציל: טראדיציאנעל, פאראויסזאגן אקציע-פרייזן אדער רעקאמענדירן ספרים און פילמען; געניראטיוו, שרייבן מעשיות, שאפן בילדער.
- באציאונג צו דאטא: טראדיציאנעלער AI ניצט דאטא צו קלאסיפיצירן אדער פאראויסזאגן; געניראטיווער AI לערנט פון דאטא צו שאפן עפעס נייעס.
- רעזולטאט: טראדיציאנעלער AI גיט לעיבלס, נומערן (scores) און קאטעגאריעס; געניראטיווער AI גיט טעקסט, בילדער, מוזיק, ווידעאס.
- ביישפילן פון סיסטעמען: טראדיציאנעל, פנים-דערקענונג, פראָד-דערקענונג, ספאם-פילטערס; געניראטיוו, E·ChatGPT, DALL, מוזיק-שאפונג.

איינפירונג צו די GenAI געצייג (1:51:52)

דער קומענדיגער קאפיטל וועט ווייזן אנווענדונגען פון E·ChatGPT, DALL און GPT-3. דער קורס הייבט אן מיט א בילד: מען ארבעט אויף א קאלעדזש-פראיעקט, ס'איז שפעט ביינאכט און דער טערמין איז מארגן; מען דארף א גוט-געשריבענעם באריכט, א פאסטער-דיזיין און א קלארע פרעזענטאציע, א גאנצער בהלה. אבער וואס אויב ס'איז דא אן AI-אסיסטענט וואס קען העלפן מיט דעם אלעם אין עטליכע מינוט? דאס איז די וועלט פון געניראטיוון AI.

עס ווערט ערקלערט ווי סטודענטן, אינהאלט-שאפער און פראפעסיאנאלן ניצן די געצייג צו טון זאכן שנעלער, גרינגער און קליגער, און מען וועט דורכגיין יעדן געצייג אינאיינעם מיט ווי צו ניצן אים אין פראקטיק. מען הייבט אן מיט ChatGPT 4: א AI-chatbot וואס קען שרייבן עסעיען, סיכומים און ארטיקלען, ערקלערן טעמעס, און לייזן קאוד-פראבלעמען.

E·ChatGPT, DALL און GPT-3 אין דער פראקטיק (1:52:55)

ChatGPT אלס א פערזענליכער לערנער (1:52:55)

דער קורס ערקלערט אז דורך איין איינציגן "טשעט" קען מען לייזן פראבלעמען, אויסלערנען מאטעריאל, און אפילו דורכפירן איבונג-אינטערוויוס. דער ChatGPT מאדעל באניצט זיך מיט GPT-4, דער לעצטער אין דער

משפחה פון GPT, וואס שטייט פאר "generative pre-trained transformer". GPT-4 איז געבויט פאר מער פיינע און מענטשליכע שמועסן.

דער ערשטער פראמפט וואס ווערט פרובירט איז: "ערקלער מיר דעם גרינהויז-עפעקט ווי איך וואלט געווען אין קלאס 12". דער פראמפט בעט א ערקלערונג, און הייסט דעם מאדעל זיך פירן ווי ער רעדט צו א תלמיד פון קלאס 12.

מען זעט אן ענטפער וואס קלינגט ווי א מענטש: דער גרינהויז-עפעקט איז א נאטירליכער פראצעס וואס ווארעמט אן די אויבערפלאך פון דער ערד; ער פאסירט ווען געוויסע גאזן אין דער ערד'ס אטמאספערע כאפן איין די היץ. דערנאך קומט א פירלייגע ערקלערונג אין פיר פונקטן: זון-ענערגיע, ווי די ערד'ס אויבערפלאך נעמט איין די ענערגיע, די גרינהויז-גאזן, און ווי די היץ ווערט איינגעכאפט. נאך די פיר פונקטן לייגט דער ענטפער צו אז אן דעם גרינהויז-עפעקט וואלט די ערד געווען צו קאלט צו האלטן לעבן, אבער מענטשליכע טעטיקייטן, ווי ברענען פאסיל-ברענשטאפן, אויסהאקן וועלדער און אינדוסטריעלע פראצעסן, פארשטארקן אים. דער קורס באמערקט אז דאס איז א קורצע ערקלערונג אין א נוסשאל, און ווער עס פארשטייט עס האט א קלארע בילד וואס דער גרינהויז-עפעקט איז.

דער ערשטער נוצן וואס מען זעט דא איז אז ChatGPT איז ווי א פריוואטער לערנער. מען גייט אריין אין chat.openai.com, מען שרייבט א פראגע, און מען באקומט א פשוט'ן ענטפער וואס מען פארשטייט גרינג. מען קען אויך פרעגן א נאכפאלג-פראגע, למשל: "איז זון-ענערגיע פארבונדן מיטן גרינהויז-עפעקט?" און דער ענטפער קומט: יא, זון-ענערגיע איז פארבונדן מיטן גרינהויז-עפעקט. דערפאר, זאגט דער קורס, איז עס גוט פאר לערנען שפעט ביי נאכט פאר אן עקזאמען, אדער ווען מען דארף שנעל פארשטיין א געדאנק.

שרייבן עסעיס און אויפגאבעס אין מינוטן (1:55:31)

דער צווייטער נוצן איז שרייבן אויפגאבעס אדער עסעיס אין עטליכע מינוטן. מען בעט: "שרייב אן עסעי פון 350 ווערטער וועגן קלימאט-ענדערונג פאר קאלעדזש". דער קורס ווייזט אז מען האט דא געגעבן באדינגונגען, און מען קען עס נאך מער צופאסן, למשל צולייגן "מאך עס פשוט אבער אינטערעסאנט".

דער מאדעל שאפט א טיטל, "א וואקסנדיקע זארג פאר אונזער צוקונפט: קלימאט-ענדערונג", און טיילט דעם טעקסט אין פאראגראפן. מען קען ענדערן די צאל ווערטער און די טעמע. דערנאך דארף מען איבערקוקן דעם געשריבענעם ענטוואורף, און מען קען אים צופאסן צו דער אייגענער פערזענליכקייט. אזוי, זאגט דער קורס, ווערט שרייבן וויניגער שטרענג און מער הנאה'דיג.

איבן פאר א דזשאב-אינטערוויו (1:56:32)

ChatGPT קען אויך געבן אן אינטערוויו-דערפארונג. דער ביישפיל: מען האט אן אינטערוויו קומענדיגן מאנטאג, און מען ווייסט נישט פונקט וואס זיך צו גרייטן. מען קען בעטן ChatGPT זיך פירן ווי אן אינטערוויוער פאר א געוויסער שטעלע. מען קען אפילו אריינקלעבן די באשרייבונג פון דער שטעלע פון יענער פירמע, און פרעגן וועלכע פראגעס מען וועט מעגליך באקומען.

אין דער דעמא שרייבט מען: "פרעג מיר דריי HR פראגעס פאר א software development internship, און שלאג פאר פארבעסערונגען". אין א רגע הייבט דער מאדעל אן שרייבן:

- ערשטע פראגע: "קענסט מיר דורכפירן א פרישן קאוד-פראיעקט וואס דו האסט געארבעט, און ערקלערן ווי דו ביסט צוגעגאנגען לייזן דעם פראבלעם?" דער קורס באמערקט אז דאס איז א זייער געוועלכע פראגע. דער מאדעל לייגט צו א עצה: די פראגע מעסט די פעאיקייט צו לייזן פראבלעמען, די טעכנישע פעאיקייט,

און ווי מען גייט צו אנטוויקלונג-ארבעט; זי גיט דעם אינטערוויוער א בליק אין דער פראקטישער דערפארונג און אין דעם דענק-פראצעס.

- צווייטע פראגע: "ווי פירסטו זיך מיט טערמינען וועו דו ארבעטסט אויף עטליכע פראיעקטן אדער אויפגאבעס אין איינעם?" די פראגע מעסט ווי מען פירט זיך מיט צייט און ווי מען שטעלט פריאריטעטן. ווייל ווייכווארג-אנטוויקלונג פאדערט אפט צו פירן עטליכע אויפגאבעס צוגלייך, גיט דאס דעם אינטערוויוער א בילד פון דער ארגאניזאציע-פעאיקייט. דער קורס באטאנט אז דאס איז זייער וויכטיג.
- דריטע פראגע: "ווי האלטסטו זיך אויפ'ן לויפנדן מיט די נייעסטע פראגראמיר-שפראכן און טעכנאלאגיעס?" די פראגע קוקט אויף דער פעאיקייט אליין צו לערנען און זיך צו פארבעסערן. דער מאדעל שרייבט אויך א ביישפיל-ענטפער דערויף.

צום סוף קומען פארשלאגן פאר פארבעסערונג: ברענג ספעציפישע ביישפילן, ווייז ארויס די "soft skills", גרייט זיך פאר פראבלעם-לייזונג פראגעס, א.א.וו.

דערנאך פאסט מען עס צו: "פאס עס צו פאר א data analyst אנהייבער אן דערפארונג; וואס קען דער אינטערוויוער מיך פרעגן אין דער טעכנישער ראונד?" און דער ענטפער קומט גלייך. HR פראגעס פאר אן אנהייבער data analyst:

- פארוואס ווילסטו א קאריערע ווי א data analyst, און וועלכע טריט האסטו געטון זיך צו גרייטן?
- קענסטו באשרייבן א פאל וועו דו האסט געדארפט אנאליזירן דאטא אדער לייזן א פראבלעם מיט ציפערן אדער דאטא, אין לערנען אדער אין א פערזענליכן פראיעקט?
- ווי גייסטו צו לערנען נייע געציג אדער טעכנאלאגיעס, ספעציעל ווייכווארג פאר דאטא-אנאליז?

טעכנישע פראגעס:

- וואס איז דער אונטערשייד צווישן סטרוקטורירטע און אומסטרוקטורירטע דאטא?
- וואס איז ?data normalization
- ווי באקוועם ביסטו מיט געצייג פאר דאטא-ויזואליזאציע? וועלכע ויזואליזאציע וואלסטו געוצט צו ווייזן פארקויף-דאטא איבער צייט?
- קענסטו ערקלערן וואס א pivot table איז, און וואו מען נוצט עס אין דאטא-אנאליז?

פארשלאגן: ווייז דיין לערן-וועג, שטעל אין צענטער די פראבלעם-לייזונג פעאיקייטן, גרייט זיך פאר די יסודות'דיגע טעכנישע פראגעס. ווייל מען איז אן אנהייבער, וועלן די טעכנישע פראגעס זיך מערסטנס קאנצענטרירן אויף די יסודות, ווי Excel, Python און איר ביבליאטעק pandas. אזוי קען מען אין א רגע מאכן אן איבונג-אינטערוויו.

אידייען פאר פראיעקטן (2:00:38)

ChatGPT קען אויך העלפן מיט אידייען, "brainstorming". דער ביישפיל: מען האט א גרופע-פראיעקט און מען דארף אן אידייע. מען שרייבט: "גיב פינף שעפערישע אידייען פאר א גרופע-פראיעקט וועגן sustainability". דער מאדעל ברענגט פינף אידייען, צווישן זיי אן ארויסרוף צו פלאנירן פאקונג וואס שאדט נישט די סביבה, און א געמיינדע-גארטן אין שטאט. ביי יעדער אידייע שרייבט ער אן ציל, א צוגאנג און א רעזולטאט. דער קורס זאגט אז אזעלכע פראיעקטן קענען זיין סיי לערנדיג סיי ווירקזאם, און מען וועט קיינמאל

נישט אנטרינען פון אינספיראציע. מען ווייזט אויך אז אין ChatGPT זענען דא נאך אפציעס, ווי זוכן, "deep search" און שאפן בילדער.

E·DALL: בילדער פון טעקסט (2:02:13)

דער קומענדיגער געצייג איז E·DALL, אן AI וואס שאפט בילדער פון טעקסט: מען זאגט וואס צו צייכענען, און עס שאפט קונסט-ווערק, לאגאס און בילדער. מען געפינט עס אינעווייניג אין ChatGPT: אין דער זייט-ליניע גייט מען צו "Explore GPTs", מען זוכט E·DALL) עס זענען דא עטליכע ווערסיעס), און מען הייבט אן שמועסן.

עס ווערט ערקלערט אז E·DALL באניצט זיך מיט א ווערסיע פון דעם CLIP מאדעל (Contrastive Language-Image Pre-training) צוואמען מיט VQ-VAE-2 אדער אנדערע מאדעלן וואס שאפן בילדער. אזוי שאפט עס זייער שעפערישע בילדער פון א באשרייבונג אין נאטירליכער שפראך.

די ביישפילן:

- א פלאקאט פאר א קאלעדזש-פעסטיוואל: "שאף א פלאקאט פאר א קאלעדזש-פעסטיוואל מיט DJs, נעאן-ליכט און א ווילדן עולם". אין א רגע קומט א שיינע בילד, און מען קען עס אראפלאדן. דער קורס זאגט: מען דארף נישט Photoshop, די AI טוט די שווערע ארבעט.
- בילדער פאר פרעזענטאציעס: "צייכן א קארטון וואס ווייזט וואסער-פארפעסטיגונג אין א טייך". די בילד קען מען אריינלייגן אין PowerPoint אדער Canva, ווי א וועקטאר, אן אייקאן, אדער וואס מען וויל.
- לאגאס פאר קלובן: "א לאגא פאר א קאווע-קלוב מיט ספרים און דעמפנדיגע טעפלעך". דער לאגא זעט אויס ווי עכט. מיט איין פראמפט באקומט מען פיר-פינף ווערסיעס, און מען קען בעטן נאך אן אנדערע ווערסיע. גוט פאר סטודענטן-קלובן און קאלעדזש-געשעענישן.
- קאסטיום-פלאנירונג: "צייכן א צוקונפטיגן קאסטיום פאר א science fiction שפיל מיט גלאנצנדיגע עלעמענטן". דאס רעזולטאט איז זייער אינטערעסאנט.

עצה: די בילדער זענען פון הויכער קוואליטעט; נוץ לעבעדיגע באשרייבונגען און מער דעטאלירטע פראמפטס, און מען וועט באקומען א זייער גוטן רעזולטאט.

GPT-3: קאוד, צוזאמענפאסונגען און איידיען (2:06:55)

דער קומענדיגער געצייג איז GPT-3, דער מאטאר הינטער געצייג ווי ChatGPT. עס שמועסט נישט נאר; עס קען שרייבן גאנצע קאוד, צוזאמענפאסן פארשונג-ארבעטן, שרייבן געשעפט-אימעילס און נאך. GPT-3 איז דער פאר-גייער פון GPT-4, אויף דער זעלבער GPT ארכיטעקטור, מיט 175 ביליאן פאראמעטערס; וועז עס איז ארויס איז עס געווען איינס פון די מערסט פארגעשריטענע שפראך-מאדעלן. אין דער דעמא נוצט מען ChatGPT) GPT-4מיט) צו ווייזן וואס GPT-3 קען.

- וועבזייטלעך אן פראגראמירן: "שרייב HTML פאר א פערזענליכן פארטפאליא מיט מיין נאמען, ביאגראפיע און א קאנטאקט-פארעם". דער קאוד קומט גלייך; מען קאפירט אים אריין אין VS Code אדער אן אנדערן קאוד-רעדאקטאר, און מען באקומט א ריכטיג רעזולטאט. גוט פאר אנהייבער אדער דיזיין-סטודענטן.
- צוזאמענפאסן א לאנגע פארשונג-ארבעט: מען קליקט אויפן "+" צייכן, מען לאדט ארויף די טעקע פונעם קאמפיוטער, און עס ווערט צוזאמענגעפאסט אין צוויי סעקונדעס.

- אידייַן פאר פראיעקטן און סטארטאפס: "שלאג פאר פינף אייגנארטיגע אפליקאציע-אידייַן פאר קאלעדזש-סטודענטן". דער ענטפער: "Study Buddy Finder", העלפט סטודענטן געפינען שותפים לויט קורסן, צייט-פלענער און פארליבונגען; "Campus Event Hub"; אויסטויש פון קורס-מאטעריאל; א "check-in" פאר גייסטיגע געזונט; "Foodie Finder", א.א.וו., מיט פונקטן פאר בעסערן פארשטאנד, ווי א שטילער לערן-טיימער. מען קען עס נוצן אנצוהייבן א hackathon אדער א קאנקורענץ.

דער קורס פירט עס נישט ווייטער, ווייל דער נוצן פון ChatGPT איז שוין געווין געווארן.

די דריי פאמיליעס פון געפֿנעראטיוו מאדעלן (2:10:03)

די קומענדיגע לעקציע איז פארשטיין געפֿנעראטיוו AI מאדעלן. דער משל: שטעל דיר פאר אז מען לערנט א קאמפיוטער צו זיין שעפעריש, ווי מאלן א בילד, שרייבן א ליד אדער פלאנירן א קלייד. דאס איז פונקט וואס געפֿנעראטיוו AI מאדעלן צילן: לערנען פון דאטא און שאפן נייעם אינהאלט וואס זעט אויס, קלינגט אדער פילט זיך ווי עכט. ווי טוען זיי דאס? מען צעטיילט זיי אין דריי הויפט-פאמיליעס:

- GANs, generative adversarial networks.
- VAEs, variational autoencoders.
- טראנספארמערס, ווי GPT.

דער ערשטער, GANs, ווערט באשריבן ווי א שפיל "דער קינסטלער קעגן דעם קריטיקער": א פארמעסט צווישן צוויי נויראל-נעצן. דער generator איז ווי א יונגער אנהייבער-קינסטלער וואס פרובירט שאפן נאכגעמאכטע קונסט; דער discriminator איז דער שטרענגער קריטיקער וואס פרובירט אונטערשיידן עכטע קונסט פון נאכגעמאכטע. דער קינסטלער שאפט צופעליגע בילדער, דער קריטיקער זאגט "נאכגעמאכט" אדער "עכט", און דער קינסטלער לערנט זיך צו ווערן בעסער.

דזשענערעטיוו מאדעלן און דער ערשטער פראקטישער מאדעל (2:11:35)

GANs: די אנווענדונגען און די שווערקייטן (2:11:35)

דער קורס ענדיגט די ערקלערונג פון GANs. ביים טריינירן ווערן די נאכגעמאכטע בילדער מיט דער צייט אזוי גוט, אז מען קען זיי שוין נישט אונטערשיידן פון די עכטע בילדער.

עס ווערט געגעבן א משל פונעם לעבן. דער generator איז ווי א תלמיד וואס פרובירט נאכצומאכן א חתימה. דער discriminator איז ווי א מלמד וואס כאפט דעם שווינדל. ווען ביידע ווערן בעסער, ווערן די נאכגעמאכטע חתימות כמעט שלימות'דיג.

די אנווענדונגען פון GANs:

- שאפן רעאליסטישע מענטשליכע פנימער.
- style transfer אין קונסט, דאס הייסט איבערטראגן א סטיל פון איין בילד אויף אן אנדער בילד.
- פראטאטיפן פאר מאדע-דיזיין.
- deepfakes. דער קורס זאגט אז דאס איז א געשטריטענע זאך, אבער טעכניש זייער אינטערעסאנט.

די שווערקייטן פון GANs:

- דאס טריינירן איז נישט סטאביל.

- מען דארף א סך דאטא און א סך קאמפיוטער-כח.

VAE: א דאטא-קאמפרעסער מיט שעפעריישקייט (2:12:35)

דערנאך קומט דער Variational Autoencoder (VAE). דער קורס באשרייבט אים ווי א "קאמפרעסער מיט שעפעריישקייט", א סיסטעם וואס פרעסט צוזאם די דאטא און בויט זי דערנאך צוריק אויף.

דער VAE נעמט איין דאטא, למשל א בילד. ער פרעסט עס צוזאם אין א קליינעם רוים מיט ווייניג דימענסיעס, וואס הייסט latent space. דערנאך בויט ער עס צוריק אויף.

דער חידוש איז אז דער VAE געדענקט נישט נאר, נאר ער "שטעלט זיך פאר". וועז מען נעמט פארשידענע פונקטן פונעם latent space, קען ער שאפן נייעם אינהאלט וואס זעט אויס ענליך צום ארגינעל.

די אנווענדונגען פון VAE:

- anomaly detection, דאס הייסט אויפכאפן א זאך וואס פאסט נישט צו די נארמאלע דאטא.
- שאפן האנטגעשריבענע ציפערן און פנימער.
- קאמפרעסירן דאטא.
- drug discovery, דאס הייסט שאפן מאלעקולארע סטרוקטורן פאר נייע מעדיצינען.

די מעלות פון VAE: ער איז מער סטאביל ווי GANs, און ער גיט א גוט-ארגאניזירטן latent space, וואס מען קען ניצן פאר interpolation, דאס הייסט פאר א גלאטן איבערגאנג צווישן פונקטן.

טראנספארמער-מאדעלן: דער מייסטער פון סעקווענצן (2:13:42)

דער דריטער מאדעל איז דער טראנספארמער. GANs און VAEs קאנצענטרירן זיך מער אויף בילדער. טראנספארמערס שיינען דערביי צו באהאנדלען סעקווענצן, בפרט טעקסט.

דער הויפט-געדאנק איז דער attention mechanism. דאס איז א וועג וואס לאזט דעם מאדעל זיך קאנצענטרירן אויף פארשידענע ווערטער וועז ער שאפט א זאץ.

דער קורס ברענגט א ביישפיל. וועז GPT שרייבט דעם זאץ "I love ice cream because it's sweet", גיט ער אכטונג אויפן קאנטעקסט. ער פארשטייט ווי דאס ווארט "because" איז פארבונדן מיט "ice cream" און מיט "sweet".

די אנווענדונגען פון טראנספארמערס:

- chatbots, ווי ChatGPT.
- שאפן קאוד, ווי GitHub Copilot.
- איבערזעצן שפראכן.
- צוזאמפאסן טעקסטן און שאפן אינהאלט.

פארוואס טראנספארמערס שטייען ארויס:

- זיי וואקסן גוט מיט גרויסע מאסן דאטא.
- מען קען זיי fine-tunen פאר ספעציפישע ארבעטן, למשל צוזאמפאסן מעדיצינישע טעקסטן אדער שרייבן לעגאלע דאקומענטן.

• זיי באהאנדלען בעסער פארבינדונגען צווישן ווערטער וואס שטייען ווייט איינס פונעם אנדערן, בעסער ווי עלטערע מאדעלן ווי RNN אדער LSTM.

א פארגלייך צווישן די דריי מאדעלן (2:15:00)

דער קורס שטעלט אוועק א טאבעלע מיט א פארגלייך:

• GANs: דער בעסטער פאר שאפן בילדער. א ביישפיל איז שאפן פנימער (face synthesis). די מעלה: הויכע רעאליזם. דער חסרון: שווער צו טריינירן.

• VAE: דער בעסטער פאר קאמפרעסירן דאטא און פאר anomaly detection. מען ניצט אים צו אויספארשן דעם latent space. די מעלה: זייער סטאביל טריינירן. דער חסרון: די רעזולטאטן זענען ווייניגער שארף.

• טראנספארמערס: דער בעסטער פאר טעקסט און שפראך. ביישפילן זענען ChatGPT און צוזאמפאסן טעקסטן. די מעלה: שטארק ווען עס איז דא א סך דאטא. דער חסרון: טייער צו טריינירן.

וועלכן מאדעל צו ניצן און ווען (2:15:54)

דער קורס ווארנט אז ווען מען ווייסט נישט וועלכן דזשענערעטיוון מאדעל צו ניצן פאר וועלכע ארבעט, קען זיך אלעס שנעל צעמישן.

א ביישפיל: מען בויט אן אפליקאציע וואס דארף שרייבן אימעילס, מאכן בילדער, ענטפערן אויף פראגעס און איבערמאכן רעדן אויף טעקסט. אויב מען ניצט דעם אומריכטיגן מאדעל פאר דער אומריכטיגער ארבעט, וועלן די רעזולטאטן זיין שוואך, אלעס וועט גיין פאמעלעך, און מען וועט פארלירן צייט.

דער כלל:

• פאר שארפע, רעאליסטישע בילדער: GANs.

• פאר וואריאציעס און אנאליז פונעם latent space: VAEs.

• פאר שפראך און טעקסט-ארבעטן: טראנספארמערס.

ווי ארבעטן די מאדעלן פון אינעווייניג (2:16:52)

דער טריינינג-פראצעס פון א GAN גייט אין דריי שריט:

• שריט 1: דער generator שאפט נאכגעמאכטע דאטא.

• שריט 2: דער discriminator פרובירט אונטערצושיידן צווישן עכט און נאכגעמאכט.

• שריט 3: מיט יעדן ראונד ווערן ביידע בעסער אין זייער ארבעט.

דער ציל צום סוף איז אפצונארן דעם discriminator מיט רעאליסטישע רעזולטאטן.

ביי טראנספארמערס ארבעטן דריי זאכן אונטער דער מאשין:

• attention mechanism: העלפט דעם מאדעל פארשטיין וועלכע ווערטער זענען מער וויכטיג.

• self-attention: לאזט דעם מאדעל קוקן אויף אלע ווערטער אין א זאץ אויף איינמאל.

• positional encoding: העלפט נאכגיין די ריי-פאלגע פון די ווערטער, ווייל טראנספארמערס לייענען נישט פון לינקס צו רעכטס ווי מענטשן.

דערפאר קענען מאדעלן ווי GPT-4 אזוי גוט צוזאמפאסן, איבערזעצן און שאפן טעקסט.

דער קורס פאסט צוזאם מיט א משל. צו פארשטיין דעם חילוק צווישן די מאדעלן איז ווי צו וויסן וועלכן געצייג צו ניצן פאר וועלכער שעפערישער ארבעט. GANs זענען די דיגיטאלע מאלער. VAEs זענען די קלוגע קאמפרעסערס מיט פאנטאזיע. טראנספארמערס זענען די "כישוף-מאכער" פון טעקסט.

שאפן דעם ערשטן דזשענערעטיוון AI מאדעל: דער פלאן (2:18:05)

אין דעם קאפיטל בויט מען דעם ערשטן דזשענערעטיוון AI מאדעל. מען טרייניררט נישט קיין מאדעל פון אנפאנג אן. אנשטאט דעם ניצט מען אדער מען fine-tunet א מאדעל וואס איז שוין געטריינירט (pre-trained). דאס מאכט די ארבעט שנעל און באקוועם פאר אנהייבער.

די שריט בקיצור:

- אויסקלייבן א pre-trained מאדעל, למשל GPT-2, ווייל ער איז לייכט און גוט פאר אנהייבער. מען נעמט אים פונעם Hugging Face Model Hub.
- צוגרייטן די סביבה אין Google Colab.
- אינסטאלירן די ביבליאטעקן transformers און torch.
- לאדן דעם מאדעל און זיין tokenizer.
- שרייבן א פראמפט און שאפן טעקסט, מיט דער pipeline API צוליב פשטות, און מיט פאראמעטערס ווי max_length און temperature.

די דעמא: אינסטאלירן און לאדן GPT-2 (2:19:25)

מען עפנט א נייעם notebook אין Google Colab און מען לויפט:

```
pip install transformers!
```

"pip install" מיינט: אינסטאלירט דעם פעקל אין Colab. transformers איז א ביבליאטעק פון Hugging Face, וואס גיט צוטריט צו שטארקע AI מאדעלן ווי GPT-2, BERT און נאך.

דערנאך שרייבט מען:

```
from transformers import pipeline

generator = pipeline("text-generation", model="gpt2")
```

עס ווערט ערקלערט אז pipeline איז א קורצער וועג. ער באהאלט דעם גאנצן קאמפליצירטן קאוד וואס מען דארף כדי צו לויפן AI מאדעלן. "text-generation" זאגט: איך וויל א מאדעל וואס קען פאר מיר שרייבן טעקסט. GPT-2 איז דער נאמען פונעם מאדעל. OpenAI האט אים געטריינירט צו טרעפן וועלכעס ווארט קומט ווייטער אין א זאץ.

מען זעט אז דער מאדעל לאדט זיך אראפ אינגאנצן, 100%.

דער ערשטער פראמפט און דער רעזולטאט (2:21:30)

מען שטעלט אוועק א פשוטן פראמפט: "Once upon a time in a world full of AI". דערנאך רופט מען דעם generator מיט דעם פראמפט, מיט max_length=100 און num_return_sequences=1. צום סוף דרוקט מען אויס result[0]["generated_text"].

די ערקלערונג פון יעדן טייל:

- דער פראמפט איז דער אנהייב-זאץ וואס מען גיט דעם AI, ווי צו זאגן: "הייב אן א מעשה מיט דעם".
- max_length=100 מיינט: שאף ביז 100 ווערטער אדער אותיות נאכן פראמפט.
- num_return_sequences=1 בעט נאר איין ווערסיע פונעם רעזולטאט. מען קען אויך בעטן צוויי אדער דריי.
- דער [0] מיט "generated_text" נעמט ארויס דעם געשאפענעם טעקסט פונעם רעזולטאט.

דער AI שרייבט ווייטער א המשך צום פראמפט. דער רעזולטאט רעדט וועגן מאנסטערס וואס מען קען היינט צום ערשטן מאל דערהרג'ענען, און וועגן נייע מאנסטערס אין א מלחמה, "in the blood of elves". דער קורס באטאנט אז דאס ווייזט דעם כח פון דזשענערעטיוו AI צו פארענדיגן א פראמפט.

קאנטראלירן די שעפערישקייט: temperature, top_k, top_p (2:24:15)

מען רופט נאכאמאל דעם generator מיט דעם זעלבן פראמפט, max_length=100 און num_return_sequences=1. דאס מאל לייגט מען צו נאך פאראמעטערס: temperature=0.7, top_k=50, top_p=0.95 און do_sample=True. דער קורס ווארנט מען זאל נישט פארגעסן די קאמע צווישן די פאראמעטערס.

די ערקלערונג:

- temperature קאנטראלירט די שעפערישקייט. א נידריגע ווערט איז "זיכער". א הויכע ווערט איז "ווילד" און שעפעריש. 0.7 איז אין מיטן.
- top_k=50 מיינט: קלייב אויס פון די 50 מערסט-מעגליכע קומענדיגע ווערטער, אנשטאט נאר פונעם ערשטן.
- top_p=0.95 איז nucleus sampling, וואס גיט א מער נאטירלעכע אומפארויסזאגבארקייט.
- do_sample=True שאלט איין די אומפארויסזאגבארקייט. אן דעם איז דער רעזולטאט צו פארויסזאגבאר.

דער רעזולטאט רעדט וועגן ווי יעדער פארשריט אין AI מאשינען האט געטוישט ווי מיר טראכטן וועגן דער וועלט. אבער דער טעקסט חזר'ט כסדר איבער "we can make sure that". יעדעס מאל וואס מען לויפט דעם קאוד קען מען באקומען א ביסל אנדערן רעזולטאט.

צום סוף פאסט דער קורס צוזאם וואס מען האט געטון. מען האט גענוצט די transformers ביבליאטעק פון Hugging Face צו לאדן GPT-2. מען האט געשריבן א פראמפט, און דער AI האט אים פארגעזעצט לויט זיין טריינירן. דאס אלעס איז געגאנגען אין Google Colab, אן קיין קאמפליצירטע סביבה פאר מאשין-לערנען.

די ווירקונג פון דזשענערעטיוו AI אין דער עכטער וועלט: געזונט-וועזן (2:27:50)

דער קומענדיגער קאפיטל הייבט אן מיט דעם געדאנק אז דזשענערעטיוו AI איז נישט נאר א מאדנע וואַרט. עס טוישט פונדאמענטאל ווי ביזנעסער ארבעטן, ווי זיי לייזן פראבלעמען, און ווי זיי באדינען קונים. AI שאפט באשרייבונגען פון פראדוקטן, און עס דיזיינט אפילו טיילן פון עראפלאנען. אזוי נעמט עס איבער ראלעס וואס האבן אמאל געפאדערט מענטשליכע שעפערישקייט און שכל.

דער ערשטער פעלד איז געזונט-וועזן, מיט שנעלערע דיאגנאזן און מיט אנטדעקן נייע מעדיצינען. דזשענערעטיוו AI העלפט דאקטוירים און וויסנשאפטלער מאכן שנעלערע באשלוסן:

- מעדיצינישע בילדער: AI קען אנאליזירן בילדער ווי X-rays אדער MRIs, און שרייבן באריכטן וואס העלפן דאקטוירים שנעל אויפכאפן א פראבלעם. א ביישפיל: א שפיטאל ניצט AI צו שרייבן באריכטן פאר די דאקטוירים, און דאס פארקירצט די צייט וואס מען פארברענגט ביי יעדן פאציענט.
- drug discovery: AI העלפט וויסנשאפטלער אנטדעקן נייע מעדיצינען שנעלער, דורכן פארויסזאגן ווי פראטעינען בייגן זיך. דאס איז וויכטיג כדי צו שאפן מעדיצינען.

אנוועגדונגען, א טשעטבאט, און די גרענעצן פון AI (2:29:30)

AI אין פינאנצן, האנדל און פאבריקאציע (2:29:30)

דער קורס ענדיגט די ריי ביישפילן ווי AI ווערט געניצט אין פארשידענע פעלדער. ביים סוף פון רפואה-פעלד ווערט דערמאנט אז AI האט געהאלפן אנטוויקלען א נייע רפואה אין בלויז 18 חדשים, א זאך וואס נעמט געווענליך פיל לענגער.

דאס צווייטע פעלד איז פינאנצן, ווו AI ברענגט קליגערע ריזיקע-פארוואלטונג און אויטאמאטישע באריכטן. עס ווערט ערקלערט אז AI מאכט גרינגער צו אנאליזירן דאטא און אויסצוכאפן פראבלעמען:

- אויטאמאטישע פינאנציעלע באריכטן: AI קען אנשרייבן פינאנציעלע באריכטן אויפן יסוד פון דאטא, און שפארט אזוי צייט פאר די אנאליסטן. למשל, א העדזש-פאנד (hedge fund) נוצט AI צו שרייבן מארקעט-באריכטן אין עטליכע מינוט.
- כאפן שווינדל (fraud detection): AI העלפט באנקען אויסגעפינען שווינדל דורכדעם וואס עס שאפט פאלשע טראנזאקציע-דאטא, מיט וועלכע מען פרובירט אויס די זיכערהייט-סיסטעמען, און מען מאכט זיי אזוי פארלעסליכער. למשל, Mastercard נוצט AI צו שאפן רעאליסטישע טראנזאקציע-דאטא כדי צו פארבעסערן איר סיסטעם פאר כאפן שווינדל.

דאס דריטע פעלד איז קליינהאנדל (retail), וואו AI שאפט פערזענליכע איינקויף-דערפארונגען פאר יעדן קונה:

- פראדוקט-באשרייבונגען: AI שרייבט באשרייבונגען צוגעפאסט פאר פארשידענע גרופעס קונים, למשל א לויזע שפראך פאר יונגע קונים און א פארמעלע פאר פראפעסיאנאלן. א קליידער-פירמע, למשל, נוצט AI צו שרייבן באשרייבונגען אויטאמאטיש פאר טויזנטער פראדוקטן.
- ווירטועלע אנפרוברונג (virtual try-on): AI לאזט קונים אנפרובירן קליידער אדער קאסמעטיק ווירטועל, צו זען ווי עס וועט אויסזען אויף זיי פאר מען קויפט. למשל, די אפליקאציע פון Sephora נוצט AI צו פארשלאגן קאסמעטיק, און דער באניצער זעט ווי עס קוקט אויס אויף זיין פנים דורך דער קאמערע פונעם טעלעפאן.

דאס פערטע פעלד איז פאבריקאציע, וואו AI מאכט שנעלער די דיזיין-ארבעט און די אויפהאלטונג פון מאשינען:

- דיזיינס געשאפן דורך AI: AI העלפט שאפן לייכטערע און שטארקערע טיילן פאר פראדוקטן, און פארמינערט אזוי אפפאל און מאטעריאל-קאסטן. למשל, א עראפלאן-פירמע האט געניצט AI צו דיזיינען טיילן וואס זענען 50% לייכטער, אבער פונקט אזוי שטארק ווי די טראדיציאנעלע דיזיינס.

• פאראויסזאגנדיגע אויפהאלטונג (AI: predictive maintenance) העלפט פאבריקן פאראויסזאגן וועז א מאשין וועט דארפן אויפהאלטונג. דאס פארמינערט די צייט וואס די מאשינען שטייען און מאכט די ארבעט מער עפעקטיוו. למשל, א פאבריק נוצט AI צו שאפן אויטאמאטיש אן אויפהאלטונג-פלאן פאר די מאשינען, לויט דער דאטא פון ווי זיי ארבעטן.

AI אין חינוך און פארווייזונג (2:32:03)

דאס פינפטע פעלד איז חינוך, וואו AI מאכט דאס לערנען מער פערזענליך און צוגענגליך:

• AI-מלמדים (tutors): א מלמד וואס ארבעט מיט AI קען העלפן תלמידים מיט זייערע היימארבעט, און זיי פירן שריט נאך שריט דורך די אויפגאבעס. למשל, א תלמיד וואס איז געבליבן שטעקן ביי א חשבון-אויפגאבע קען באקומען אנווייזונגען און ערקלערונגען פון אן AI-מלמד, וואס איז צוגעגרייט צו יעדער צייט.

• אויטאמאטישער אפרוף (feedback): לערער נוצן AI צו שאפן קוויזן און צו געבן באמערקונגען אויף אויפגאבעס, און שפארן אזוי שעהן יעדע וואך. למשל, א לערער נוצט AI צו דורכקוקן עסייען און צו געבן באמערקונגען אין עטליכע מינוט אנשטאט שעהן.

דאס זעקסטע פעלד איז פארווייזונג, וואו AI שאפט אינהאלט אין גרויסע מאסן, שנעלער און ביליגער:

• שרייבן סקריפטן: AI קען העלפן שרייבער אויפקומען מיט אידייען, שאפן דיאלאגן, און אפילו מאכן "סטאריבאורדס" (story boards) פארן פילמירן. למשל, א פילם-מאכער נוצט AI צו פלאנירן די סצענעס און שאפן בילדער פאר מען הייבט אן פילמירן, און שפארט אזוי צייט אין דער צוגרייטונג-שטאפל.

• ווירטועלע פערזאנען: AI קען שאפן ווירטועלע פיגורן וואס רעדן און טוען, וואס איז נוצליך פאר פילמען, שפילן און אינהאלט פאר סאציאלע מעדיע. למשל, מען נוצט AI צו שאפן ווירטועלע "אינפלוענסערס" וואס האבן שייכות מיט זייערע נאכפאלגער אויף סאציאלע מעדיע.

וואס איז די Hugging Face Transformers ביבליאטעק (2:33:34)

דערנאך ווייזט דער קורס ווי אזוי מען בויט א טשעטבאט אין Google Colab. פאר מען גייט צום קאוד ווערט ערקלערט וואס די Hugging Face Transformers ביבליאטעק איז. עס איז איינע פון די פאפולערסטע ביבליאטעקן פאר נאטירליכע שפראך-באארבעטונג (natural language processing). זי שטעלט צו פאר-טריינירטע מאדעלן וואס קענען טון א ריי ארבעטן, ווי למשל שאפן טעקסט, סארטירן טעקסט, אויסגעפינען נעמען (named entity recognition), און נאך.

אין קאוד נוצט מען א מאדעל וואס הייסט DialoGPT-small. דאס איז א נוסח פון GPT-2 וואס איז געמאכט פאר שאפן שמועסן. DialoGPT איז געווען טריינירט אויף גרויסע דאטא-סעטס פון שמועסן, און דערפאר פאסט עס זייער גוט פאר בויען טשעטבאטס.

צוגרייטן די סביבה און ארייננעמען די ביבליאטעקן (2:34:34)

דער ערשטער שריט איז צו אינסטאלירן די ביבליאטעק מיטן באפעל:

```
pip install transformers
```

די שורה אינסטאלירט די transformers ביבליאטעק. מען דארף זי כדי צו ארבעטן מיט די פאר-טריינירטע מאדעלן וואס Hugging Face שטעלט צו. זי דערמעגליכט צו לאדן מאדעלן ווי DialoGPT, אן AI-מאדעל פאר שמועסן, און עס נוצן צו שאפן ענטפערס אינעם טשאט.

דערנאך נעמט מען אריין די נויטיגע ביבליאטעקן:

```
from transformers import AutoModelForCausalLM, AutoTokenizer

import torch
```

עס ווערט ערקלערט וואס יעדער טייל טוט:

- AutoModelForCausalLM: די קלאס לאדט א פאר-טריינירטן מאדעל פאר "causal language modeling", וואס ווערט געוצט פאר שאפן טעקסט. מיט איר לאדט מען דעם DialoGPT מאדעל.
- AutoTokenizer: דער טאקענייזער פארוואנדלט דעם טעקסט אין "טאקענס", דאס הייסט נומערן וואס דער מאדעל קען באארבעטן. נאכדעם פארוואנדלט ער דעם רעזולטאט פונעם מאדעל צוריק אין טעקסט וואס א מענטש קען לייענען.
- torch: דאס איז PyTorch, א דיפ-לערנען פריימווערק וואס דער מאדעל נוצט. מען נוצט עס פאר ארבעטן מיט טענזארן (tensors), ווייל די מאדעלן פארלאזן זיך אויף טענזארן סיי פאר דעם אריינגאב און סיי פאר דעם ארויסגאב.

לאדן דעם מאדעל און אנהייבן דעם טשאט (2:36:07)

דער נעקסטער שריט איז צו לאדן דעם מאדעל און דעם טאקענייזער. דער טאקענייזער פארוואנדלט דעם רויען טעקסט, דאס הייסט וואס דער באניצער שרייבט אריין, אין טאקענס וואס דער מאדעל פארשטייט. יעדעס ווארט אדער פראזע באקומט אן אייגענעם ID-נומער, און דאס מאכט גרינגער פארן מאדעל צו באארבעטן און צו שאפן אן ענטפער. דערנאך לאדט מען דעם פאר-טריינירטן DialoGPT-small מאדעל. ער איז געווען טריינירט אויף א גרויסע זאמלונג פון דיאלאגן, און דערפאר קען ער שאפן פאסיגע ענטפערס אין א שמועס-שטייגער.

ביים לויפן די צעל זעט מען אז די טעקע tokenizer_config.json ווערט אראפגעלאדן ביז 100%. עס ווערט אנגעוויזן אז א גוטע זאך ביי Colab איז אז עס ווייזט דעם סטאטוס פון יעדע טעקע וואס ווערט אראפגעלאדן, ווי למשל merges.txt און vocab.json, ווי ווייט זי איז פארטיג, און צי עס איז דא א פראבלעם.

דערנאך הייבט מען אן דעם טשאט:

- chat_history_ids: א וואריאבל ווא מען היט אפ די היסטאריע פונעם שמועס. דאס העלפט דעם מאדעל צו האלטן דעם קאנטעקסט איבער עטליכע אריינגאבן פונעם באניצער.
- א print-באפעל וואס שרייבט אויס א באגריסונג, אז מען זאל אנהייבן שמועסן מיט דעם AI, און אז מען קען שרייבן quit כדי ארויסצוגיין. אזוי באגריסט מען דעם באניצער און מען זאגט אים דעם באפעל ווי ארויסצוגיין.

דער הויפט-לופ פונעם טשעטבאט (2:37:40)

דער הויפט-לאגיק פונעם טשעטבאט איז א לופ. מען שרייבט for step in range(5), און אינעווייניג נעמט מען דעם אריינגאב פונעם באניצער מיט input, מיטן צייכן "User: " אלס הינווייז. דערנאך שרייבט מען: if

user_input.lower() == "quit": break, דאס הייסט אז אויב דער באניצער שרייבט quit, ווערט דער לופ אפגעשטעלט און דער טשאט ענדיגט זיך.

אין דער דעמא שרייבט מען ערשט "hey", און דער לופ האלט ווייטער. נאכדעם שרייבט מען quit, און מען זעט אז עס הערט אויף, דאס הייסט עס איז ארויס פונעם לופ.

דערנאך קומט דאס קאדירן פונעם אריינגאב. דער טעקסט פונעם באניצער ווערט פארוואנדלט אין טאקען-IDs מיט tokenizer.encode. אינעם סוף פונעם אריינגאב לייגט מען צו דעם EOS טאקען (end of sequence). ער זאגט דעם מאדעל אז דער אריינגאב האט זיך געענדיגט, און אז ער זאל זיך גרייטן צו שאפן אן ענטפער. דער טייל "pt"return_tensors= זארגט אז דער קאדירטער אריינגאב קומט צוריק אלס א PyTorch טענזאר, וואס דאס איז דער פארמאט וואס דער מאדעל ערווארט.

פארוואלטן די היסטאריע, שאפן און אויסלייגן דעם ענטפער (2:39:45)

דער ציל פון פארוואלטן די היסטאריע פונעם שמועס איז צו האלטן דעם קאנטעקסט, כדי דער מאדעל זאל קענען ענטפערן פאסיג צום שמועס. אויב עס איז דא פריערדיגע היסטאריע, דאס הייסט chat_history_ids, ווערט זי צוגעהעפט (concatenated) צום נייעם אריינגאב, און אזוי נעמט דער מאדעל אין באטראכט דעם גאנצן שמועס ביז יעצט. אויב עס איז נישטא קיין היסטאריע, דאס הייסט עס איז דער ערשטער אריינגאב, נוצט מען גלייך די נייע input_ids.

דערנאך שאפט מען דעם ענטפער:

```
model.generate(bot_input_ids, max_length=1000, = chat_history_ids
pad_token_id=tokenizer.eos_token_id)
```

- model.generate שיקט דעם צוגעהעפטן אריינגאב צום מאדעל און שאפט אן ענטפער.
- max_length באשטימט די מאקסימאלע לענג פונעם געשאפענעם טעקסט, דא 1000 טאקענס.
- pad_token_id זארגט אז די "פעדינג" טאקענס ווערן באהאנדלט ווי געהעריג, ווייל דער מאדעל דארף וויסן וועלכע טאקענס זענען בלויז פילעכץ.

דערנאך לייגט מען אויס (decode) דעם ענטפער: response = (...)tokenizer.decode, וואס נעמט דעם טייל פון chat_history_ids וואס הייבט זיך אן נאך דער לענג פון bot_input_ids, דאס הייסט נאר די נייע טאקענס. אזוי ווערן די געשאפענע טאקען-IDs פארוואנדלט צוריק אין טעקסט וואס א מענטש קען לייענען. דער פאראמעטער skip_special_tokens=True זארגט אז ספעציעלע טאקענס, ווי דער EOS, זאלן נישט ארויסקומען אינעם ענדגילטיגן רעזולטאט. צום סוף דרוקט מען אויס דעם ענטפער מיט print, און דער באניצער זעט אים.

ווי אזוי עס ארבעט, און פארוואס א טשעטבאט (2:41:47)

עס ווערט צוזאמגעפאסט וואס עס טוט זיך ווען מען לויפט דעם קאוד. מען הייבט אן דעם טשאט, און דער באט באגריסט אייך אז איר זאלט אנהייבן שמועסן. יעדעס מאל וואס איר שרייבט עפעס אריין, נעמט דער באט אייער מעסעדזש, קאדירט אים, און שיקט אים צום מאדעל צוזאמען מיט דער היסטאריע פונעם שמועס. דער מאדעל שאפט אן ענטפער וואס נעמט אין באטראכט דעם גאנצן שמועס ביז יעצט, און דער באט ווייזט דעם ענטפער. דאס גייט אזוי ווייטער ביז איר שרייבט quit.

עס ווערט ערקלערט פארוואס עס איז וויכטיג צו בויען א טשעטבאט:

• ער פארגרעסערט דעם באניצער'ס אינטערעס, ווייל ער גיט פערזענליכע ענטפערס גלייך אויפן ארט. דאס איז מער לעבעדיג און באקוועם ווי סטאטישער אינהאלט.

• ער איז צוגעגרייט 24/7, אזוי אז באניצער באקומען גלייך אן ענטפער צו יעדער צייט.

• ער טוט אויטאמאטיש איבער-חזר'דיגע ארבעטן, ווי ענטפערן אויף געווענליכע פראגעס, באארבעטן באשטעלונגען, אדער געבן גרונטלעכע קונה-שטיצע. אזוי באפרייט ער ווערטפולע רעסורסן, און די מענטשלעכע ארבעטער קענען זיך קאנצענטרירן אויף מער קאמפליצירטע ארבעטן.

• ער פארבעסערט די צופרידנקייט פון די קונים, ווייל ער לייזט שנעל זייערע פראגעס, פארקירצט דאס ווארטן, און פארבעסערט די גאנצע דערפארונג פונעם באניצער.

• מיט פארוואלטונג פון קאנטעקסט געדענקט דער באט פריערדיגע שמועסן, און גיט אזוי א המשך'דיגע און מער פערזענליכע דערפארונג.

• מיט באהאנדלונג פון שמועסן מיט עטליכע ריינדן (multi-turn) קען דער באט פירן אהין-און-צוריק שמועסן, אנשטאט נאר ענטפערן אויף איינצלנע פראגעס. אזוי פילט ער זיך מער ווי א מענטשלעכער שמועס.

אלע די אייגנשאפטן צוזאמען שאפן א קלוגן, עפעקטיוון טשעטבאט וואס פארבעסערט שטארק דעם קאנטאקט מיט באניצער און מאכט גלאטער די ביזנעס-אפעראציעס. עס ווערט אויך פארגעשלאגן אז מען קען "פיין-טונען" (fine-tune) א מאדעל מיט אן אייגענעם דאטא-סעט, כדי ער זאל זיין מער צוגעפאסט צו א געוויסן פעלד אדער באנוץ. מען קען אים אויך צוגרייטן צו באהאנדלן לענגערע שמועסן, אזוי אז ער זאל געדענקען די היסטאריע פונעם טשאט פאר א לענגערע צייט.

די גרענעצן פון געניראטיווע AI מאדעלן (2:44:24)

די נעקסטע לעקציע באהאנדלט די באגרענעצונגען פון געניראטיווע AI מאדעלן:

• מאנגל אין פארשטאנד: AI מאדעלן שאפן אינהאלט דורך פאראויסזאגן וואס קומט ווייטער, לויט די מוסטערן אין דער דאטא, אבער זיי פארשטייען נישט באמת דעם אינהאלט. למשל, ווען AI שרייבט אן ארטיקל אדער קאוד, ווייסט עס נישט וואס עס שרייבט; עס מישט נאר צוזאם מוסטערן לויט דער פריערדיגער דאטא.

• אפהענגיגקייט אין דער קוואליטעט פון דער דאטא: AI איז נאר אזוי גוט ווי די דאטא אויף וועלכער עס איז טריינירט געווארן. אויב די דאטא איז פארטייאיש אדער נישט פולשטענדיג, קען עס שאפן פארטייאישע אדער אומגענויע רעזולטאטן. עס ווערט געבראכט די ביישפילן פון פארטייאישע אלגאריטמען ביים אננעמען ארבעטער, און פון פעלערהאפטע רפואה-עצות. דאס זענען אמת'ע פראבלעמען וואס קומען פון שלעכטער טריינירונג-דאטא.

• דער דריטער פונקט, וואס ווערט נאר אנגעהויבן אינעם סוף פון דעם אפשניט, איז באגרענעצונגען אין שעפערישקייט.

ווייטערדיגע חסרונות פון געניראטיווע AI (2:45:24)

דער קורס פירט ווייטער מיט דער רשימה פון די שוואכקייטן פון געניראטיווע AI. דער ערשטער פונקט אין דעם שטיק איז שעפערישקייט: AI קען ארויסגעבן גאר שיינע שעפערישע זאכן, למשל קונסט-ווערק אדער מוזיק. אבער עס האט שוועריגקייטן צו ברענגען עפעס באמת נייעס און אריגינעלס, אזוי ווי א מענטשליכער קינסטלער. עס ווערט ערקלערט אז AI נעמט בעיקר אינהאלט וואס עקזיסטירט שוין און מישט אים איבער, אנשטאט צו דערפינדן עפעס וואס ברעכט נייע וועגן.

דערנאך קומען די עטישע זארגן. AI-מאדעלן קענען אמאל ארויסגעבן שעדליכן אינהאלט. דאס גייט פון באליידיגנדיגע שפראך ביז "deep fakes", וואס מען קען זיי שוועער אונטערשיידן פון אמת'ע פילם-בילדער. דאס שאפט א סכנה פון פאלשע אינפארמאציע און עטישע פראבלעמען, בפרט אין מעדיע און פארוויילונג.

דער דריטער פונקט איז קאנטעקסט און דקות. געניראטיווע AI קען פארפעלן די פיינע נואנסן אין א שמועס אדער אין געפילן. א ביישפיל: אויב מען בעט דעם מאדעל הילף אין א שפירעוודיגן ענין, ווי גייסטיגע געזונט, קען ער געבן אן עצה וואס איז אלגעמיין, און נישט צוגעפאסט צו די געפילישע באדערפענישן פון דעם יחיד.

ווייטער: הוצאות און ריסורסן. טריינירן א גענעראטיוון AI-מאדעל פאדערט ריזיגע מאסן קאמפיוטער-קראפט, צייט און ענערגיע. די מאדעלן לויפן געוועלדיך אויף שטארקע סערווערס וואס פארצערן אסאך ריסורסן. דאס הייבט אויף שאלות וועגן ווי לאנג מען קען דאס אויסהאלטן, און וועגן דער השפעה אויף דער סביבה.

דערנאך: באגרעניצטע פעאיגקייט צו לייזן פראבלעמען. געניראטיווע AI איז אויסגעצייכנט אין ארבעטן ווי שרייבן עסעיס אדער צוואמנעמען טעקסט. אבער עס פאסט נישט פאר ארבעטן וואס פאדערן טיפערע סברא אדער קאמפליצירטע פראבלעם-לייזונג. עס קען "שפילן" ווי עס לייזט א פראבלעם, אבער עס פארשטייט נישט באמת די לייזונג ווי א מענטש.

דער נעקסטער פונקט איז די סכנה פון זיך צופיל פארלאזן. וויבאלד AI שאפט מער און מער אינהאלט, קען מען ווערן צו אָפהענגיג אויף דער טעכנאלאגיע. מענטשן קענען זיך פארלאזן אויף AI פאר ארבעטן וואס פאדערן מענטשליכע עקספערטיז אדער קריטיש טראכטן. דאס קען שטערן די פערזענליכע שעפערישקייט און דאס באשליסן.

ווייטער: פראבלעמען מיט פריוואטקייט פון דאטא. AI-מאדעלן קענען אמאל שאפן אינהאלט באזירט אויף פערזענליכע דאטא, וואס ברענגט זארגן וועגן פריוואטקייט. דאס איז ספעציעל א פראבלעם אין געביטן ווי געזונט-פלעגע, וואו שפירעוודיגע דאטא קען אומגעוואוסט ווערן געטיילט אדער מיסברויכט.

דער לעצטער חסרון איז די שוועריגקייט צו אלגעמיינערן. געניראטיווע מאדעלן זענען גוט אין ספעציפישע ארבעטן, אבער האבן שוועריגקייטן מיט א בריייטערע אלגעמיינערונג. א ביישפיל: א מאדעל וואס איז טריינירט געווארן צו שאפן בילדער פון קעץ, קען האבן שוועריגקייטן צו שאפן איבערצייגנדיגע בילדער פון א גאנץ אנדער חיה. דאס ווייזט אז AI איז באגרעניצט אין זיין בייגעוודיגקייט.

די צוקונפט פון GenAI-מאדעלן (2:48:30)

דערנאך גייט דער קורס איבער צו דער צוקונפט פון GenAI-מאדעלן, אין צען פונקטן:

• בעסערע פארשטאנד: אין דער צוקונפט וועלן מאדעלן ווארשיינליך פארשטיין דעם קאנטעקסט טיפער, און געבן מער פאסיגע און באדייטפולע תוצאות. עס קענען קומען מאדעלן וואס פארשטייען באמת דעם טייטש הינטער די ענטפערס, אנשטאט נאר ארויסצושפייען דאטא.

• פארשטארקטע שעפערישקייט: וועט די מאדעלן אנטוויקלען זיך, קען מען ערווארטן אז זיי וועלן שטופן די גרעניצן און ווערן מער שעפעריש. דורך פארשריטן אין טעכניקן ווי reinforcement learning און neural architecture search, קען AI אנהייבן שאפן באמת אריגינעלן אינהאלט. דאס קען מאכן א רעוואלוציע אין געביטן ווי קונסט, מוזיק און ליטעראטור.

• מער עטישע AI: מען וועט ווארשיינליך זען מער רעגולאציעס ארום דעם באנוץ פון AI, כדי צו פארמיידן שעדליכע תוצאות. AI-פירמעס ארבעטן שוין צו פארבעסערן די דורכזיכטיגקייט, צו פארמינערן פארורטיילן (bias), און צו מאכן זייערע מאדעלן מער עטיש און פראנטווארטליך.

• בעסערע פערזענליכקייט: AI וועט קענען געבן א מער צוגעפאסטע, פערזענליכע איבערלעבונג. מען זאל זיך פארשטעלן אן AI וואס פארשטייט באמת אייערע באדערפענישן און וואס איר האט ליב. עס וועט אייך געבן אינהאלט, פראדוקטן אדער סערוויסעס וואס פילן זיך פערזענליך און פאסיג.

• ענערגיע-שפארעוודיגקייט: וויבאלד די פראגע נאך געגנראטיוע AI וואקסט, קאנצענטרירן זיך פארשער אויף שאפן מאדעלן וואס פארברויכן ווייניגער ענערגיע. מען קען זען קלענערע אבער שטארקערע מאדעלן וואס לויפן מיט ווייניגער קאמפיוטער-קראפט, און פארמינערן אזוי די השפעה אויף דער סביבה.

• צוזאמענארבעט צווישן מענטש און AI: אנשטאט צו פארביטן מענטשן, וועט די צוקונפט ווארשיינליך זיין געבויט אויף צוזאמענארבעט. AI וועט העלפן מענטשן זיין מער פראדוקטיוו, פארשטארקן זייער שעפעריישקייט, און געבן איינזיכטן וואס מענטשן קענען נוצן צו מאכן בעסערע באשלוסן.

• אינטעגראציע מיט נייע טעכנאלאגיעס: געגנראטיוע AI וועט זיך ווייטער פארבינדן מיט נייע טעכנאלאגיעס ווי augmented reality (AR), virtual reality (VR) און blockchain. דאס עפנט נייע מעגליכקייטן פאר "אריינטונקענדיגע" איבערלעבונגען און פאר דעצענטראליזירטע אפליקאציעס.

• לייזן קאמפליצירטע פראבלעמען פון דער וועלט: אין דער נאענטער צוקונפט קען מען זען AI-מאדעלן ארבעטן צוזאמען מיט וויסנשאפטלער אויף גרויסע אויפגאבעס, ווי קלימאט-ענדערונג, מגפות, און די זיכערקייט פון שפייז אויף דער וועלט. AI קען העלפן אנאליזירן דאטא, פאראויסזאגן טרענדס, און פארשלאגן נייע לייזונגען.

• מער צוטריטלעכע AI: וועט די טעכנאלאגיע גייט פאראויס, וועט AI ווערן מער צוטריטלעך און די געצייג וועלן זיין גרינגער צו באנוצן. מענטשן פון אלע שיכטן, נישט נאר טעכנישע מומחים, וועלן קענען אויסנוצן די קראפט פון AI צו שאפן און צו דערפינדן.

דער קורס שליסט דעם חלק מיט א פאזיציע: כאטש געגנראטיוע AI-מאדעלן אנטוויקלען זיך נאך, קען מען נישט אפלייקענען זייער פאטענציאל צו איבערמאכן אינדוסטריעס, צו פארשטארקן שעפעריישקייט, און צו לייזן קאמפליצירטע פראבלעמען. פונדעסטוועגן איז וויכטיג זיך צו באשעפטיגן מיט זייערע באגרעניצונגען, און צו ארבעטן אויף מער עטישע, שפארעוודיגע און פראנטווארטליכע AI-סיסטעמען.

אין א קורצן איבערבליק וועלן דערמאנט וואס מען האט געלערנט אין דעם גאנצן חלק: וואס געגנראטיוע AI איז, ווי עס האט געווירקט אויף דער וועלט, בויען chatbots, שאפן דעם ערשטן GenAI-מאדעל, און די צוקונפט פון GenAI-מאדעלן. די עצה איז צו פרובירן ווייטער מיט אייגענע דאטא-סעטן, און צו פארבעסערן דעם אייגענעם "pipeline", כדי צו בויען קליגערע סיסטעמען וואס פארשטייען דעם קאנטעקסט.

ChatGPT אלס געהילף פאר דאטא-אנאליז (2:53:12)

א נייער חלק הייבט זיך אן: ווי אזוי ChatGPT קען פארגרינגערן דאטא-אנאליז. דער פראבלעם וואס ווערט באשריבן איז ווען מען קען נישט ארויסקריגן באדייטפולע איינזיכטן פון דער דאטא. עס ווערט ערקלערט אז ChatGPT קען העלפן אנטדעקן מוסטערן, טרענדס און איינזיכטן, סיי ביי קליינע דאטא-סעטן און סיי ביי קאמפליצירטע, אן צעמישענישׂ און אן דרוק.

מען קען עס באטראכטן ווי א קלוגן געהילף. עס קען צוזאמנעמען דאטא, ענטפערן שאלות וועגן טרענדס, און אפילו פארשלאגן איינזיכטן וואס מען קען גלייך אויספירן, אלעס אין א פשוט'ער, פארשטענדליכער שפראך. דאס פאסט סיי פאר אן אנהייבער און סיי פאר אן ערפארענעם אנאליטיקער.

ChatGPT ארבעט אזוי: עס טייטשט אויס וואס מען גיט אריין, און גיט קלארע ענטפערס. מען קען פרעגן שאלות וועגן דער דאטא, בעטן קלארקייט אויף קאמפליצירטע פונקטן, און אויסטראכטן אידייען פאר באשלוסן באזירט אויף דאטא. אזוי נעמט עס ארויס די קאמפליצירטקייט פון דער אנאליז, און מען קען זיך קאנצענטרירן אויף דעם עיקר: פארשטיין די דאטא און נוצן די איינזיכטן.

די נייע פעאיגקייטן און וואס איז דאטא-אנאליז (2:54:45)

אין דעם דעמא ווערן קודם געוויזן די פארבעסערונגען אין דאטא-אנאליז ביי ChatGPT:

- מען קען ארבעטן מיט טאבעלעס און טשארטס אין א נייעם אויסגעברייטערטן בליק.
- מען קען צולייגן טעקעס גלייך פון Google Drive אדער OneDrive, און אזוי ארויפלאדן די לעצטע ווערסיע פון א טעקע.
- מען קען צופאסן און אראפלאדן טשארטס פאר פרעזענטאציעס און דאקומענטאציע, וואס איז א זייער גוטע זאך.

די פארבעסערונגען זענען צוטריטלעך אין דעם נייעם הויפט-מאדעל GPT-4o פאר באניצער פון ChatGPT Plus, און פאר Team- און Enterprise-באניצער אין די קומענדיגע וואכן.

דערנאך ווערט ערקלערט וואס דאטא-אנאליז איז. עס איז א פראצעס און א געצייג וואס גיט ארגאניזאציעס די קראפט צו מאכן אינפארמירטע באשלוסן, צו פאראויסזאגן טרענדס, און צו פארבעסערן די עפעקטיווקייט פון זייער ארבעט. עס איז דער רוקנביין פון סטראטעגישן פלאנירן אין ביזנעס, רעגירונג און אנדערע ארגאניזאציעס. ChatGPT העלפט שפארן צייט אין געפינען מוסטערן און טרענדס, אין עטליכע סעקונדעס, נישט אמאל עטליכע מינוט.

מען קען אויסקלייבן דעם ספעציעלן "Data Analyst" אינסטרומענט אין ChatGPT . אדער, ווער עס האט ChatGPT Plus, קען ארבעטן גלייך מיט GPT-4o, און אין דעם דעמא נעמט מען GPT-4o.

דעמא: ארויפלאדן און באשרייבן א Netflix דאטא-סעט (2:56:17)

מען קען ארויפלאדן א דאטא-סעט אויף צוויי ווענען: גלייך פונעם קאמפיוטער, אדער פון Google Drive . ביי Google Drive דארף מען קודם פארבינדן דעם אקאונט. אין דעם דעמא איז די דאטא נישט געווען אין Google Drive, דעריבער לאדט מען ארויף א טעקע פון Netflix גלייך פונעם קאמפיוטער.

דער ערשטער פראמפט איז "describe the data". ChatGPT אנאליזירט די טעקע און באשרייבט זי. דער דאטא-סעט אנטהאלט אינפארמאציע ווענען Netflix שאוס און פילמען: 12 קאלומען און בערך 8,800 רייען. דער דאטא-סעט איז אראפגעלאדן געווארן פון Kaggle, און מען קען אים אויך דארט אראפלאדן. צווישן די

קאלומען זענען show_id, type, title, director, cast, country, duration און description. ChatGPT ווייזט אויך א מוסטער פון די דאטא, למשל א פילם פון USA מיט א רעיטינג פון PG-13, א דאקומענטאר וואס שפיגלט אפ דאס לעבן פונעם פאטער פון דעם פילם-מאכער. ChatGPT פארשלאגט שוין אליין ווייטערדיגע שריט, למשל צו ווייזן ווי די דאטא איז צעטיילט לויט פילם-סארט.

דעמא: רייניגן די דאטא (2:58:18)

פאר מען גייט ווייטער, בעט מען "please clean this data". ChatGPT מעלדט אז דער דאטא-סעט איז גערייניגט געווארן מיט די פאלגנדע ענדערונגען:

- אויסגעמעקט דופליקאטן, כדי יעדער רעקארד זאל זיין אייגנארטיג.
- באהאנדלט פעלנדע ווערטן: אין די קאלומען director, cast און country האט מען זיי אויסגעפילט מיט "Unknown"; ביי דער דאטום פון צולייגן האט מען אריינגעשטעלט "Not Specified"; און ביי rating "Not Rated".

דערנאך ווייזט ChatGPT א פארנייטע צוואמענפאסונג פון דעם דאטא-סעט: די קאלומען זענען נישט געענדערט, עס זענען נישט געבליבן קיין דופליקאטן, און נישט קיין קריטישע פעלנדע ווערטן.

דעמא: פינף טשארטס, דער Python קאוד, און א pie chart (2:58:50)

דער נעקסטער פראמפט: "please give me five different visual charts from the data". מען דערמאנט נאכאמאל אז מען קען אויך נוצן דעם Data Analyst אינסטרומענט, אבער מיט דער Plus ווערסיע ארבעט מען גלייך מיט GPT-4o. ChatGPT שאפט די פאלגנדע פינף טשארטס:

- די צאל פילמען קעגן TV שאוס.
- די 10 לענדער מיט דעם מערסטן אינהאלט.
- די 10 דירעקטארן מיט די מערסטע טיטלען; אויבן איז "Unknown", און דערנאך Rajiv Chilaka.
- די צאל אינהאלט לויט רעיטינג.
- די צאל טיטלען וואס זענען צוגעלייגט געווארן יעדעס יאר.

ביי יעדן טשארט קען מען איבערגיין צו אן אינטעראקטיוון טשארט, אים אראפלאדן, אדער אים אויסברייטערן אויף א גרעסערן בליק. מען זעט אויך די "legend". אזוי, ווערט ערקלערט, קען מען אין עטליכע סעקונדעס פאראויסזאגן טרענדס און שאפן פארשידענע סארטן טשארטס.

דערנאך נוצט מען איינעם פון די פארגעשלאגענע פראמפטס: צו געפינען פאפולערן אינהאלט לויט דעם יאר וואס ער איז ארויסגעקומען. וען מען קליקט אויף "View analysis", זעט מען דעם Python קאוד וואס שטייט הינטער דעם טשארט. מען קען אויך אויסקלייבן אז דער פולער פרט זאל שטענדיג ווערן געוויזן. מען קען דעם קאוד קאפירן און אריינלייגן אין אן אייגענעם Python IDE, און דארט וועט ער ארויסגעבן דעם זעלבן גראף וואס GPT האט געשאפן.

צום סוף בעט מען א pie chart לויט לאנד. עס ווערט אויך דערמאנט אז מען קען שפעטער מאכן א היסטאגראם און א wordcloud. דער טשארט ווייזט די צעטיילונג פון אינהאלט לויט די 10 לענדער מיט דעם מערסטן אינהאלט: United States מיט 2,818 טיטלען, India מיט 972, און "Unknown" מיט 831.

וואָרט-וואלקן און טרענדס פון דעם דאטא-סעט (3:02:04)

דער אנהייב פון דעם אפשניט ענדיגט א דעמא פון אנאליזירן א דאטא-סעט פון פילמען מיט ChatGPT. מען זעט א טשארט מיט צאלן ווי "Kingdom" און 419 245, און דער קורס באטאנט ווי שנעל דאס איז געגאנגען: בלויז פיר-פינף סעקונדעס. מען קען אויסברייטערן די אנזיכט און זיך באקוקן דעם טשארט פון נאענט. עס ווערט ערקלערט אז דאס איז די קראפט פון ChatGPT: עס פארקירצט שטארק די צייט וואס מען דארף צו אנאליזירן דאטא. מען קען אויך זען דעם קאוד וואס שטייט הינטער יעדן טשארט.

דערנאך ווערט געבעטן: "please create word cloud" פון דעם דאטא-סעט. ChatGPT שאפט א וואָרט-וואלקן (wordcloud), און מען זעט דארט ווערטער ווי family, woman, life, take, new, help, team, father. דערנאך ווערט געבעטן א וואָרט-וואלקן לויט די נעמען פון די פילמען. עס שטעלט זיך ארויס אז אין דעם דאטא-סעט הייסט די קאלום נישט "movie" נאר "title", און ChatGPT געפינט דאס אליין און שאפט א נייעם וואָרט-וואלקן מיט ווערטער פון די טיטלען, למשל black, wall, people. דער קורס באמערקט אז מען קען דאס טוישן ווי מען ווייל. מען קען אויך בעטן פראגעס ווי "show trends of movie genre globally", צו זען טרענדס פון פילם-זשאנערס אויף דער גאנצער וועלט.

אלע אנאליזן אין א פרעזענטאציע (3:03:40)

דער נעקסטער שריט: וואס טוט מען אז מען וויל אריינענעמען די גאנצע אנאליז אין איין פרעזענטאציע, כדי עס צו ווייזן פאר מיטארבעטער אדער שותפים (?stakeholders מען שרייבט צו ChatGPT: "please give me all this analysis in the presentation format". ChatGPT מעלדט אז די אנאליז איז צוזאמגעשטעלט געווארן אין א PowerPoint-פרעזענטאציע, און עס גיט א לינק צו אראפלאדן.

מען לאדט אראפ דעם טעקע, מען עפנט עס, און מען דריקט "enable editing" כדי מען זאל עס קענען רעדאגירן. אלע אנאליזן געפינען זיך שוין אין דער פרעזענטאציע. דערנאך ווייזט דער קורס ווי מען קען באנוצן דעם איינגעבויטן "Designer" אין PowerPoint: מען דריקט אויף Designer, און עס ווערן געשאפן פארשידענע מוסטערן פאר יעדן סלייד. מען קלייבט אויס איין אויסזען פאר איין סלייד, און פאר יעדן ווייטערדיגן סלייד א צווייטן אויסזען, ביז די גאנצע פרעזענטאציע זעט אויס שיין און פראפעסיאנעל.

א נייע לעקציע: לערנען Python מיט ChatGPT (3:05:17)

דא הייבט זיך אן א נייער טייל פון דעם קורס: א רייזע אין דער וועלט פון פראגראמירן מיט דער הילף פון איינעם פון די מערסט אנטוויקלטע AI מאדעלן, ChatGPT, כדי צו לערנען Python. עס ווערט ערקלערט אז צי מען איז א גאנצער אנהייבער צי מען וויל פארברייטערן די פראגראמיר-קענטענישן, איז Python א געצילטע ברירה, ווייל עס איז באקאנט פאר זיין פשטות, זיין ביגוואמקייט (versatility) און די ברייטע ריי באניצונגען. מיט דער הילף פון ChatGPT ווערט דאס לערנען לעבעדיג און אינטעראקטיוו.

עפענען ChatGPT און זיך איינשרייבן (3:05:48)

מען גייט צום בראוזער. מען קען זוכן "openai" אדער "chat gpt" און דריקן אויף דעם לינק פון ChatGPT. דער אפיציעלער וועבזייטל איז openai.com/chatgpt, און דארט דריקט מען "get started". עס ווערט באמערקט אז ChatGPT איז אויך פארהאן אויף Google Play Store און Apple Store.

א נייער באניצער מוז זיך רעגיסטרירן; וועט עס האט שוין א חשבון, קען זיך גלייך אריינלאגן. מען קען זיך אויך רעגיסטרירן מיט א Google-חשבון. נאכן רעגיסטרירן קומט מען צו דער שמועס-זייט (chat), וואו מען הייבט אן דעם שמועס מיט ChatGPT. דארט קען מען באקומען א פולע מאפע (road map) ווי אזוי צו וועלן א Python-פראגראמירער, אדער א מאפע פאר א ספעציפישן פעלד, למשל ווי צו וועלן א דאטא-וויסנשאפטלער.

"וואס איז Python?" (3:06:48)

די ערשטע פראגע צו ChatGPT איז: "what is Python". ChatGPT ענטפערט אז Python איז א הויך-ניוואָ, אינטערפרעטירטע פראגראמיר-שפראך, באקאנט פאר פשטות, לייענבארקייט און ביגזאמקייט. עס איז געשאפן געווארן דורך Guido van Rossum און צום ערשטן מאל ארויסגעגעבן אין 1991.

ChatGPT רעכנט אויך אויס די הויפט-אייגנשאפטן:

- לייענבארע סינטאקס – Python באנוצט איינרוקונג (indentation, ליידיגע פלאץ) צו באשטימען קאוד-בלאקן.
- אן אינטערפרעטירטע שפראך.
- דינאמישע טיפן (dynamic typing).
- א רייכע סטאנדארט-ביבליאטעק.
- ארבעט אויף אלע פלאטפארמעס (cross-platform).
- אביעקט-אריענטירט.
- א ברייטע קהילה און אן אויסגעברייטערטע סביבה (ecosystem).

ChatGPT אלס לערער: א מאפע פאר Python (3:07:51)

דערנאך בעט מען ChatGPT ער זאל זיך אויפפירן ווי א לערער: "kindly act as an instructor and give me a road map or teach me Python". דער קורס באמערקט אז דא האט מען צוזאמגעמישט צוויי פארלאנגען; מען קען באנוצן איינס: "kindly act as an instructor and teach me Python", אדער "give me a road map for a Python developer". מען קען אויך אנגעבן א צייט-ראם, למשל 6 חדשים אדער 2 וואכן, ווי מען וויל.

ChatGPT ענטפערט מיט פרייד און גיט א שריט-ביי-שריט מאפע: צום אנהייב – אינסטאלירן Python און אויסקלייבן א קאוד-רעדאקטאר. דערנאך די יסודות: וועריאבלס און דאטא-טיפן, אפעראטארן, קאנטראל-סטרוקטורן, ליסטעס און דיקשאנעריס, פונקציעס און מאדולן. דערנאך מיטלסטע באגריפן, אנטוויקלטע טעמעס, וועב-אנטוויקלונג און פראיעקטן, צוזאמארבעט און וועלסיע-קאנטראל, וועלטער אנטוויקלטע טעמעס, און צום סוף – פראקטיצירן און לערנען וועלטער.

א פלאן פאר פיר חדשים (3:09:27)

דערנאך בעט מען: "kindly act as our guide and give us a complete plan to become Python developer in 4 months". ChatGPT ענטפערט אז עס וועט גערן העלפן מאכן א פולשטענדיגן פלאן פאר פיר חדשים:

- חודש איינס – יסודות און באזיס-פראגראמירן: וואכן 1–2 אריינפיר אין Python; וואכן 3–4 פונקציעס און מאדולן.

• חודש צוויי – מיטלסטע Python און אביעקט-אריענטירט פראגראמירן: וואכן 2–1 דאטא-סטרוקטורן; וואכן 4–3 אביעקט-אריענטירט פראגראמירן (OOP).

אזוי האט ChatGPT צוגעגרייט א פולן וועגווייזער פאר פיר חדשים, און מען קען אים נאכפאלגן.

מקורות צום לערנען (3:10:31)

מען פרעגט: "can you provide resources from where can I learn?" ChatGPT גיט א רשימה:

• אנליין לערן-פלאטפארמעס: Codecademy, Coursera, Udemy, edX.

• אינטעראקטיווע פראקטיק: LeetCode, HackerRank.

• ביכער: "Python Crash Course" פון Eric Matthes, און "Automate the Boring Stuff with Python".

• די אפיציעלע דאקומענטאציע – דער קורס באטאנט אז דאס איז די וויכטיגסטע זאך צו לערנען סיי וועלכע שפראך.

• YouTube-קאנאלן, בלאגס און וועבזייטלעך.

מען קען אויך בעטן אז ChatGPT זאל אריינגענעמען די מקורות אין דעם וועגווייזער אדער אין דער פולער מאפע, און עס וועט דאס טון.

צעטיילן דעם פלאן: וואך איינס-צוויי (3:11:34)

דערנאך פרעגט מען: "can you elaborate the concepts I should learn in week one to two". עס ווערט ערקלערט אז ChatGPT פירט ווייטער דעם זעלבן שמועס – עס געדענקט וואס מען האט פריער געפרעגט, און מען קען אים ווייטער אויספרעגן. עס ווייסט שוין פון וועלכע "וואך איינס-צוויי" מען רעדט. עס ענטפערט אז אין די ערשטע צוויי וואכן וועט מען פאקוסירן אויף בויען א פעסטן יסוד אין Python.

מען קען אויך אנהייבן נאר מיט וואך איינס. דארט שטייט: פארשטיין וואס Python איז און זיין ראלע אין פראגראמירן; אינסטאלירן Python אויפן קאמפיוטער און איינשטעלן א קאוד-רעדאקטאר אדער IDE , למשל Visual Studio Code אדער Jupyter Notebook, ווי מען וויל; און שרייבן דעם ערשטן פראגראם – דעם קלאסישן "hello world". טעג דריי און פיר – ווע‌ריאבלס און דאטא-טיפן.

ערקלערן א באגריף פאר אן אנהייבער (3:13:08)

עס ווערט געוויזן ווי מען באנוצט ChatGPT ווען מען בלייבט שטעקן. מען פרעגט: "can you explain the topic variables and data types to a beginner". ChatGPT ענטפערט אז עס וועט צעלייגן די באגריפן אויף א פשוטן אופן: א ווע‌ריאבל איז ווי א כלי (container) וואס האלט אינפארמאציע, ווי א קעסטל מיט א צעטל, וואו מען קען אריינלייגן פארשידענע זאכן. אין פראגראמירן באנוצט מען ווע‌ריאבלס צו האלטן און פירן די דאטא וואס דער פראגראם דארף. עס ווערט באמערקט אז ChatGPT האט געגעבן גוטע ביישפילן און באדעקט אלע אינפארמאציע וועגן ווע‌ריאבלס און דאטא-טיפן.

"ערקלער עס פאר א זעקס-יעריג קינד" (3:14:42)

א וויכטיגער טיפ: אויב מען האט נישט פארשטאנען דעם באגריף נאכן לייענען, קען מען בעטן אז ChatGPT זאל עס נאך מער פארפשוטן. מען קאפירט די פראגע און שרייבט: "can you explain the topic to a six-year-old", דאס הייסט, ערקלערן פאר א יונג קינד, מיט ביישפילן.

ChatGPT ענטפערט: שטעל דיר פאר דו האסט א ספעציעלן קעסטל וואס הייסט ווערייאבל; דאס קעסטל קען האלטן פארשידענע זאכן – צאלן, ווערטער און נאך. א ווערייאבל איז ווי א סלאוי מיט א צעטל, ווען מען האלט זאכן – מארמלשטיינדלעך (marbles), צוקערקעס, וואס מען וויל. דער סלאוי האט א נאמען, למשל "magic jar", און ווען מען וויל באנוצן וואס איז דערין, זאגט מען בלויז דעם נאמען פונעם סלאוי. דאטא-טיפן: ס'זענען פארהאן פארשידענע סארטן זאכן וואס מען קען אריינלייגן אין סלאוי – מארמלשטיינדלעך, צוקערקעס און סטיקערס; אזוי אויך אין פראגראמירן זענען פארהאן פארשידענע סארטן זאכן וואס מען קען אריינלייגן אין א ווערייאבל.

די גאנצע מעטאדע אין קורצן (3:15:43)

דער קורס פאסט צוזאם: ערשט באקומט מען א מאפע, און מען בעט ChatGPT זיך אויפצופירן ווי א לערער אדער וועגווייזער. דערנאך צעטיילט מען די מאפע – למשל, וואס צו לערנען אין וואך איינס-צוויי, דערנאך נאר וואך איינס, און דערנאך טאג איינס-צוויי. מען קען פרעגן וואו צו געפינען מקורות, ווי אזוי אראפצולאדן און איינשטעלן Python און אן IDE, און מען פאלגט די מקורות. אויב מען טרעפט א שווערigkייט, קלעבט מען אריין די פראבלעם אין ChatGPT און עס לייזט עס. אזוי גייט מען ווייטער צו וואך דריי-פיר. אויב מען פארשטייט נישט, בעט מען פארפשוטן – מען לייגט צו צום סוף פון דער פראגע "פאר א זעקס-יעריג קינד" אדער "פאר אן אנהייבער", און עס וועט ערקלערן מיט ביישפילן.

א פשוטער קאוד פאר צוזאמלייגן צאלן (3:16:45)

צום סוף פרעגט מען: "can you write a simple addition of numbers code in Python and explain the code to me". ChatGPT גיט א פשוטן Python-קאוד וואס לייגט צוזאם צוויי צאלן: איין ווערייאבל באקומט די ערשטע צאל דורך input פונעם באניצער, א צווייטער באקומט די צווייטע צאל, אין א ווערייאבל פאר דעם רעזולטאט (sum) לייגט מען זיי צוזאם, און דערנאך דרוקט מען עס אויס מיט print . דערנאך צעלייגט ChatGPT דעם קאוד שריט ביי שריט, און ערקלערט אז די ערשטע שורה בעט דעם באניצער אריצושרייבן א צאל.

פון Python און Java מיט ChatGPT ביז LLM-ס (3:18:20)

ענדיגן Python לערנען מיט ChatGPT (3:18:20)

דער קורס ענדיגט דעם טייל וועגן לערנען Python מיט ChatGPT. מען זעט ווי ChatGPT ערקלערט א קורצן קאוד וואס בעט דעם באניצער אריינצוגעבן א נומער. עס ערקלערט אז די פונקציע input נעמט דאס וואס דער באניצער טייפט אריין און היט עס אפ אין א ווריאבל. די פונקציע int פארוואנדלט דעם אינפוט אין א גאנצע נומער (integer). אזוי קען מען לערנען וואס יעדע שורה קאוד מיינט, וואס יעדע פונקציע טוט און ווי מען באניצט זי, און אזוי כאפט מען די שפראך.

מען קען ווייטער בעטן ChatGPT ערקלערן נאך באגריפן, למשל: "גיב מיר ביישפילן פאר list, tuple און dictionaries". מען קען גיין לויט דעם קורס אדער דעם וועגווייזער וואס עס האט געגעבן.

מען קען אויך פרעגן וועגן קליינע פראיעקטן. דער קורס קערט זיך אום צום "ראוד-מעפ" וואס מען האט פריער באקומען, וואו דער דריטער חודש הייסט "אדוואנסירטע טעמעס און פראקטישע אנווענדונגען". מען קען בעטן ChatGPT צעטיילן און פארפשוטן יעדן טייל פונעם קורס אדער דעם גאנצן ראוד-מעפ.

א ביישפיל: מען בעט "provide a detailed complete road map for data science". ChatGPT הייבט אן אויסלייגן דעם גאנצן ראוד-מעפ. דערנאך קען מען פרעגן וועלכע באגריפן מען דארף לערנען אין דער ערשטער אדער צווייטער וואך, און דערנאך בעטן: "ביטע פיר מיך דורך אלע באגריפן מיט ביישפילן". אזוי ווערט ChatGPT א שטארקער געצייג אויף דעם גאנצן וועג פון לערנען Python. דער קורס דערמאנט אז פראגראמירן לערנען איז א וועג און נישט א ציל, און Python איז א גוטער "וואגן" אויף דעם וועג.

לערנען Java מיט ChatGPT: א פלאן פון חדשים ביז טעג (3:20:52)

דער ווייטערדיגער טייל ווייזט ווי מען לערנט Java מיט ChatGPT. דער קורס פארגלייכט עס צו האבן א פריינטלעכן מלמד בייזייט, וואס ערקלערט אלעס גרינג, פון די יסודות ביז אדוואנסירטע באגריפן. מען לערנט שרייבן קאוד שריט נאך שריט, אנגעהויבן פון פשוטע זאכן ווי וואריאבלן ביז קאמפליצירטערע פראגראמען. מען באקומט איבונגען און ביישפילן. עס איז ווי א שמועס: מען פרעגט, און ChatGPT העלפט טרעפן דעם ענטפער. סיי א אנהייבער סיי א פראפעסיאנאל קען אזוי לייזן פראבלעמען שנעלער און לערנען נייע זאכן.

ערשטער שריט: מען שרייבט אין ChatGPT: "גיב מיר א פלאן לויט חדשים צו לערנען Java פון אנהייבער ביז אדוואנסירט". דער ענטפער:

- חודש 1: אנהייבן מיט Java יסודות: אריינפיר צו Java, און די OOP באגריפן (אביעקט-אריענטירטע פראגראמירונג).
- חודש 2: Collection Frameworks, exception handling, און file input/output.
- חודש 3: אדוואנסירטע Java טעמעס.
- חודש 4: אדוואנסירטע Java באגריפן און frameworks, Java frameworks, וועב-אנטוויקלונג מיט Java.
- חודש 5: אדוואנסירטע טעמעס און פראיעקט-ארבעט.

דער קורס באמערקט אז אזא פלאן קוקט אויס א ביסל שוואר. דעריבער שרייבט מען: "באניץ דעם פלאן פון אויבן און גיב מיר א פלאן לויט וואכן". אין דער ערשטער וואך: אריינפיר צו Java און איינשטעלן די אנטוויקלונג-סביבה. מען דארף אינסטאלירן JDK (Java Development Kit) מיט אן IDE, און דערנאך אנהייבן מיט וואריאבלן, דאטא-טיפן און אפעראטארן. דערנאך קומען וואך 2, וואך 3, וואך 4 א.א.וו.

נאכדעם קאפירט מען און שרייבט: "באניץ דעם וואכן-פלאן פון אויבן און גיב מיר א פלאן לויט טעג". איצט באקומט מען טאג 1, טאג 2, טאג 3, טאג 4 א.א.וו., וואס איז גרינגער זיך אויף צו קאנצענטרירן. אין דעם ביישפיל קומט אויס א פלאן פון 27 טעג, און מען קען אים נאכפאלגן.

ערקלערן די באגריפן פון יעדן טאג (3:24:26)

מען בעט: "ערקלער קורץ די באגריפן פון טאג 1". ChatGPT ערקלערט אז אויף טאג 1 שטעלט מען איין די אנטוויקלונג-סביבה: וואס מיינט איינשטעלן די סביבה, וואס איז JDK און וואס עס אנטהאלט, וואס איז אן IDE (integrated development environment), און ווי אזוי מען אינסטאלירט JDK.

דערנאך בעט מען טאג 2 (אפילו מיט א טעות אין אויסלייג, פארשטייט ChatGPT). אויף טאג 2: די געשיכטע פון Java, די באזיס-סינטאקס, וואריאבלן און דאטא-טיפן. וואריאבלן ווערן באניצט צו האלטן דאטא אין א פראגראם, און זיי קענען האלטן פארשידענע סארטן ווערטן.

טאג 3 באהאנדלט control flow statements (if, switch). מען שרייבט גלייך: "ערקלער מיר control flow statements מיט א ביישפיל". ChatGPT ערקלערט אז אזעלכע באפעלן קאנטראלירן דעם פלוס פון

אויספירונג אין א פראגראם לויט געוויסע תנאים. עס זענען דא דריי סארטן: if-else, switch statement , און loop statements. פאר יעדן ווערט געגעבן די סינטאקס און א ביישפיל, מיט באמערקונגען (comments) אינעם קאוד פאר בעסערן פארשטאנד. ביי loops ווייזט עס יעדן סארט לופ באזונדער. דער קורס באטאנט ווי גרינג עס איז אזוי: מען פרעגט וואס מען וויל און באקומט א ענטפער.

נאך א ביישפיל פון OOP: "ערקלער classes און objects אין Java מיט א ביישפיל". ChatGPT ווייזט די סינטאקס פון א class, א ביישפיל מיט א קלאס MyClass, און ערקלערט אז אן object איז אן instance פון א class , באשאפן לויט דעם "בלו-פרינט" וואס דער קלאס דעפינירט. אזוי קען מען לערנען יעדע פונקציע אין Java: טייפט מען איר נאמען און בעט מען אן ערקלערונג.

פראקטיצירן מיט פראגראמירן-פראגעס און פראיעקטן (3:27:42)

נאך די באגריפן גייט מען איבער צו איבונגען, ווי געפינען פרים-נומערן אדער די Fibonacci-סעריע. מען שרייבט: "גיב מיר קאוד צו געפינען פרים-נומערן אין Java". ChatGPT גיט דעם קאוד. דערנאך שרייבט מען: "ערקלער מיר דעם קאוד", און עס צעטיילט אים שריט נאך שריט: דאס איז די main מעטאד פונעם פראגראם; עס דעפינירט ערשט די ווריאבלן lower bound און upper bound, א.א.וו.

דערנאך: "גיב מיר קאוד פאר די Fibonacci-סעריע אין Java", און ווידער קען מען בעטן אן ערקלערונג. דער קורס זאגט אז אזוי קען מען לערנען יעדע פראגראמירונג-שפראך מיט ChatGPT.

צום סוף, פראיעקטן. מען שרייבט: "באזירט אויף די טעמעס וואס מיר האבן געלערנט, גיב מיר פארשלאגן פאר Java פראיעקטן". פארשלאגן:

- א פשוטע קאלקולאטאר-אפליקאציע.
- א סיסטעם צו פירן סטודענטן (student management system).
- א סיסטעם צו פירן סחורה (inventory management system).

אזעלכע פראיעקטן קען מען צולייגן צו דער ליסטע פראיעקטן, אויף GitHub, אדער אינעם רעזומע. מען קלייבט אויס א To-Do ליסטע און בעט: "גיב מיר קאוד פונעם To-Do list פראיעקט". ChatGPT גיט א פשוטע אימפלעמענטאציע אין Java מיט JavaFX פאר דעם גראפישן באניצער-אינטערפעיס. אזוי קען מען באשאפן יעדן פראיעקט און בעטן ChatGPT ער זאל עס ערקלערן.

וואס זענען גרויסע שפראך-מאדעלן (3:29:49)

דער קורס גייט איבער צו א נייע טעמע: large language models (LLM). וואס געשעט הינטער די קוליסן וועז מען טייפט עפעס אריין אין ?ChatGPT אסאך מענטשן מיינען אז דאס איז א זייער קלוגע זוך-מאשין, אבער דאס איז נישט גאנץ ריכטיג. LLM-ס זענען סטאטיסטישע פאראויסזאג-מאשינען: געגעבן א סעריע ווערטער, זאגן זיי פאראויס וועלכעס וואלט קומט ווייטער. דאס קלינגט צו פשוט, אבער וועז מען טריינירט א מאדעל אויף כמעט דעם גאנצן אינטערנעט, ביליאנען ספרים, ארטיקלען, קאוד און ווסנשאפטלעכע ארבעטן, לערנט דער מאדעל נישט בלויז ווערטער נאר באגריפן, באציאונגען און מוסטערן פון שכל-לאגיק.

א LLM איז א נויראל-נעץ, געווענליך מיט ביליאנען פאראמעטערס, טריינירט צו פארשטיין און שאפן מענטשלעכע שפראך. GPT-2 האט געהאט 1.5 ביליאן פאראמעטערס, GPT-3 איז געשפרונגען צו 175 ביליאן, און די מאדעלן וואס ווערן היינט טריינירט האבן הונדערטער ביליאנען, מעגליך טריליאנען. יעדער

פאראמעטער איז ווי א קליינער קנעפל, און טריינירן מיינט צו פארדרייען אלע ביז דער מאדעל ווערט זייער גוט אין פאראויסזאגן שפראך.

די אנטוויקלונג פון שפראך-פראצעסירונג (3:31:19)

אין די ערשטע טעג פון natural language processing זענען געווען כללים-באזירטע סיסטעמען: פראגראמירער האבן מיט דער האנט געשריבן כללים, ווי "אויב דער זאץ אנטהאלט X, ענטפער Y". דערנאך זענען געקומען סטאטיסטישע מעטאדן: bag of words מאדעלן, n-grams, און Naive Bayes קלאסיפיצירער. די אמת'ע אנטוויקלונג איז געקומען מיט דיפ-לערנען: אין 2013 האט Word2Vec געוויזן אז מען קען פארשטעלן ווערטער ווי וועקטארן. אין 2017 האט איין ארבעט אלעס געטוישט, "Attention Is All You Need", וואו Google פארשער האבן אריינגעפירט די transformer ארכיטעקטור, דער יסוד פון יעדן גרויסן שפראך-מאדעל היינט.

נויראל-נעצן און back propagation (3:31:50)

דער ערשטער יסוד-באגריף איז דער נויראל-נעץ, ערקלערט אויך פאר די וואס האבן נאך קיינמאל נישט געהערט דערפון. א נויראל-נעץ איז לויז אינספירירט פונעם מוח: עס זענען דא שיכטן פון נקודות, גערופן neurons. די דאטא קומט אריין דורך דער input שיכט, ווערט פארוואנדלט דורך די hidden שיכטן, און שאפט דעם output. יעדע פארבינדונג צווישן neurons האט א) weightוואג). טריינירן מיינט צו פארריכטן די weights אז דער אויסגאנג פונעם נעץ זאל קומען נענטער און נענטער צום ריכטיגן ענטפער.

דער פראצעס הייסט back propagation: דער מאדעל מאכט א פאראויסזאג, פארגלייכט עס מיטן ריכטיגן ענטפער, רעכנט אויס דעם טעות, און שיקט דעם טעות צוריק דורכן נעץ צו דערהיינטיגן די weights. וועז מען טוט דאס אסאך מאל אויף ריזיגע דאטא-סעטס, באקומט מען א מאדעל וואס לערנט טאקע.

transformers און דער חסרון פון RNN (3:32:51)

פאר די transformers איז די הויפט-ארכיטעקטור פאר שפראך געווען RNN (recurrent neural networks), וואס פראצעסירן טעקסט סדר'דיג, ווארט נאך ווארט. דער חסרון: ביז זיי דערגרייכן דעם סוף פונעם זאץ, האבן זיי שווער צו געדענקען וואס איז געווען אין אנהייב. transformers האבן דאס אינגאנצן אויסגעמעקט.

דער ביישפיל: "The cat, which was chased by the dog and hid under the table, was hungry." וועז א RNN לייענט דעם זאץ, ביז ער קומט צום ווארט "hungry" האט ער שוין פארגעסן וואס דער זאץ רעדט, צי "cat" איז געווען יחיד אדער רבים, און אז "cat" איז דער סובייעקט. דעריבער קען ער פאראויסזאגן "were" אנשטאט "was". דאס איז דער חסרון פון RNN.

טראנספארמערס, LLM-ארכיטעקטור און א ערשטער פראיעקט (3:33:54)

דער attention-מעכאניזם און transfer learning (3:33:54)

דער קורס ערקלערט אז ביי עלטערע מאדעלן ווי RNN, וואס לייענען די ווערטער איינס נאכן אנדערן, וועז עס זענען פאראן אסאך ווערטער אינצווישן, קאנצענטרירט זיך דער מאדעל נאר אויפן נעענטערן נאַון, ווייל דאס ווואָרט שטייט אין סוף פונעם זאץ. די טראנספארמערס מעקן אויס דעם פראבלעם אינגאנצן: אנשטאט צו גיין

סעקוואנציעל, קוקן זיי אויף אלע ווערטער גלייכצייטיג, און זיי ניצן דעם attention mechanism כדי אויסצורעכענען אויף וועלכע אנדערע ווערטער יעדעס וואָרט דארף אכטונג געבן.

א ביישפיל: "The trophy didn't fit in the suitcase because it was too big" (דער טראפעי איז נישט אריין אין טשעמאדאן ווייל ער איז געווען צו גרויס). א מענטש פארשטייט גלייך אז "it" מיינט דעם טראפעי און נישט דעם טשעמאדאן. דער attention-מעכאניזם לאזט דעם מאדעל מאכן די זעלבע פארבינדונג: ער לערנט זיך צייכענען באציאונגען צווישן ווערטער וואס זענען ווייט איינס פונעם אנדערן, און כאפט אזוי דעם קאנטעקסט אויף אן אופן וואס RNN האט פשוט נישט געקענט.

מאטעמאטיש רעכנט attention אויס פאר יעדעס וואָרט דריי זאכן: א query, א key און א value. מען קען עס פארגלייכן מיט א search engine: דער query איז וואס מען זוכט, די keys זענען וואס איז פאראן, און די values זענען וואס מען קען טאקע ארויסנעמען.

דערנאך ווערט ערקלערט אז טריינירן א גרויסן מאדעל פון אנהייב נעמט וואכן אויף טויזנטער GPUs און קאסט מיליאנען דאלאר. דערפאר: וואס אויב מען קען נעמען א מאדעל וואס איז שוין געטריינירט אויף א ריזיגן אלגעמיינעם דאטא-סעט, און אים fine-tune'ן פאר א ספעציפישער ארבעט מיט א קליינעם חלק פונעם רעכן-כח? דאס איז transfer learning, און דערפאר קען מען נעמען אן open source מאדעל ווי Llama און אין איין טאג אים פארוואנדלען אין א ספעציאליזירטן געהילף.

די רייזע פון טעקסט דורכן LLM (3:35:24)

דער קורס בויט א מענטאלן בילד פון וואס עס טוט זיך ווען טעקסט גייט אריין אין א large language model:

- ערשטער שריט – מען שיקט אריין דעם אינפוט-טעקסט.
- tokenization – דער טעקסט גייט נישט אריין ווי ווערטער נאר ווי tokens. א token איז בערך דריי ביז פיר אותיות אין דורכשניט. דאס וואָרט "unhappiness" קען ווערן דריי tokens: "un", "happi", "ness". פארוואס tokens און נישט ווערטער? ווייל אזוי קען דער מאדעל באהאנדלען סיי וועלכעס וואָרט, דורך עס צעטיילן אין שטיקלעך וואס ער דערקענט.
- embedding – יעדער token ווערט פארוואנדלט אין א vector מיט אסאך דימענסיעס, געווענליך הונדערטער אדער טויזנטער נומערן.
- transformer layers – די embeddings גייען דורך עטליכע transformer-בלאקן; מאדערנע מאדעלן האבן פון 12 ביז 96 אזעלכע שיכטן. יעדער בלאק האט צוויי הויפט-טיילן: א multi-head attention שיכט און א feed forward network. multi-head attention מיינט אז דער מאדעל לויפט attention עטליכע מאל פאראלעל; יעדער "head" לערנט זיך אכטונג צו געבן אויף אן אנדער סארט באציאונג – איינער קען זיך קאנצענטרירן אויף סינטאקס, און אן אנדערער אויף ווייטע פארבינדונגען (long range dependencies).
- positional encoding – ווייל טראנספארמערס קוקן אויף אלע tokens גלייכצייטיג, ווייסן זיי נישט די סדר פון די ווערטער. "dog bites man" און "man bites dog" וואלטן אויסגעזען אידענטיש אן א וועג צו באצייכענען די פאזיציע. דער ארגינעלער טראנספארמער האט צוגעלייגט צו יעדן token-embedding א positional encoding vector – א מאטעמאטישע חתימה באזירט אויף דער פאזיציע אין דער סעקווענץ. ער האט גענוצט sine און cosine כוואליעס אין פארשידענע פרעקווענצן, אזוי אז יעדע פאזיציע באקומט א יינציגארטיגן מוסטער וואס דער מאדעל קען פארשטיין. נייערע מאדעלן ווי GPT ניצן learned positional

embeddings, און די לעצטע ארכיטעקטורן ניצן rotary position embedding, וואס באהאנדלט זייער לאנגע סעקווענצן אסאך בעסער.

• output – דער לעצטער שריט: מען זאגט פאָרויס דעם קומענדיגן token.

די "גרויסע דריי" ארכיטעקטורן (3:36:58)

• GPT – א decoder-only טראנספארמער. ער איז autoregressive: געפערירט טעקסט איין token אויפאמאל, פון לינקס קיין רעכטס, יעדער נייער token באזירט אויף אלעס פריער. גוט פאר טעקסט-געפערירן, chatbots און שפעריישן שרייבן.

• BERT – אן אנדער צוגאנג: אן encoder אפטימירט פאר פארשטיין טעקסט. ער קוקט אויפן גאנצן זאץ bidirectional – זעט קאנטעקסט סיי פון לינקס סיי פון רעכטס. פאסיג פאר classification, ענטפערן פראגעס און information retrieval. Google ניצט BERT כדי באדייטנד צו פארבעסערן די קוואליטעט פון זוכן.

• T5 – encoder-decoder: קאמבינירט אן encoder וואס לייענט און פארשטייט דעם אינפוט, מיט א decoder וואס געפערירט דעם אויסגאנג. מען ניצט עס פאר טראנספארמאציע-ארבעטן: איבערזעצונג, סיכומים און ענטפערן פראגעס.

pre-training און דער טריינירונג-לופ (3:38:01)

pre-training איז די ערשטע פאזע: דער מאדעל לערנט זיך פון רויען טעקסט מיט א self-supervised ציל, געווענליך next token prediction. דער לופ ארבעט אזוי:

• מען לאדנט א batch פון token-סעקווענצן;

• forward pass – מען באקומט פאָרויסזאגונגען;

• מען רעכנט אויס דעם loss – ווי פאלש די פאָרויסזאגונגען זענען געווען;

• backpropagation – מען רעכנט אויס די gradients;

• optimizer step – מען דערהיינטיגט די weights, און מען חזר'ט דעם פראצעס n מאל.

LLMs אין דער פראקטישער וועלט (3:38:32)

• chatbots – ניצן AI צו נאכמאכן מענטשלעכע שמועסן און העלפן באניצער אין רעאלער צייט. context management לאזט זיי געדענקען פריערדיגע שמועסן פאר בעסערע ענטפערס. כלים ווי LangChain און LlamaIndex העלפן פארבינדן דאטא-מקורות און פארבעסערן דאס ריזאנירן.

• קאוד-געפערירן – GitHub Copilot, autocomplete און טעסטן. AI-כלים קענען שרייבן קאוד, פארשלאגן פארענדיגונגען, און העלפן פראגראמירער שרייבן שנעלער; זיי העלפן אויך געפינען טעותים, שרייבן test cases און debug'ן. דאס הייבט אויף די פראדוקטיוויטעט און פארמינערט האנט-ארבעט.

• דאקומענט-אנאליז – AI קען שנעל פראצעסירן גרויסע דאקומענטן און ארויסנעמען די הויפט-אינפארמאציע אין סעקונדעס: סיכומים, אויפכאפן וויכטיגע סעיפים, און אנאליזירן דאטא; שפארט צייט קעגן לייענען מיט די הענט.

• APIs – API integration לאזן פראגראמירער אריינלייגן AI אין אפליקאציעס אן בויען מאדעלן פון אנהייב. hosted סערוויסעס ווי OpenAI און Anthropic גיבן גרייטע מאדעלן, וואס מאכט די אנטוויקלונג שנעלער און scalable.

• self-hosting – לויפן מאדעלן אויף אייגענער אינפראסטרוקטור אנשטאט cloud. כלים ווי Ollama און vLLM גיבן בעסערע קאנטראל, פריוואטקייט און צופאסונג; נוצליך פאר ארגאניזאציעס מיט סענסיטיווע דאטא.

• on-device AI – לויפט גלייך אויפן טעלעפאן אדער לאקאלן מכשיר אן אינטערנעט. frameworks ווי Core ML און ML Kit ערמעגליכן שנעלערע פראצעסירונג און בעסערע פריוואטקייט; געווענליך געוצט פאר פנים-דערקענונג און קול-דערקענונג.

דער ערשטער LLM פראיעקט: א קליינער GPT-2 (3:40:38)

אין דער לעקציע ווערט געוויזן ווי צו בויען א ערשטן LLM פראיעקט. א ווארענונג: מען זאל טוישן דעם runtime צו T4 GPU; אויף CPU וועט עס נעמען בערך 5–4 שעה, אויף T4 GPU – 10 ביז 15 מינוט. די שריט:

• מען אינסטאלירט datasets און transformers.

• מען לאדנט דעם GPT-2 tokenizer, וואס פארוואנדלט רויען טעקסט אין נומערן וואס דער מאדעל פארשטייט.

• מען בויט א dataset-קלאס: א אייגענער קלאס וואס לייענט א טעקסט-טעקע, tokenize'ט זי, און האקט זי אויף אין שטיקער פון 128 tokens; יעדעס שטיק ווערט איין טריינירונג-ביישפיל.

• מען בויט דעם מאדעל: א קלענערע ווערסיע פון GPT-2 מיט 6 שיכטן און 256 דימענסיעס – לייכט גענוג צו טריינירן אויף פרייע הארדווער, אבער פארט א עכטער ארבעטנדיגער טראנספארמער.

• מען דאונלאודט דעם WikiText דאטא-סעט: ריינע Wikipedia ארטיקלען, געשפייכלט ווי א טעקסט-טעקע פון וועלכער דער dataset-קלאס לייענט.

• training arguments – ווי די טריינירונג זאל לויפן: batch size, learning rate, אויטאמאטישע GPU-דערקענונג.

• טריינירן – דער trainer לויפט דעם גאנצן לופ; מען זעט ווי דער loss פאלט יעדע 50 שריט – אזוי לערנט זיך דער מאדעל די שפראך-מוסטערן.

נאכן טריינירן טעסט מען: מען אימפארטירט pipeline פון transformers , און מען לאזט גענעריירן א זאץ מיט max 100 tokens, אנהייבנדיג מיט "history of artificial intelligence". דער רעזולטאט הייבט טאקע אן מיט די ווערטער, גייט ווייטער מיט "in December 2015…" און ענדיגט נאך 100 tokens – א סימן אז די טרייניירונג איז געלונגען.

דער פראבלעם, און fine-tuning מיט LoRA (3:43:45)

דערנאך גיט מען דעם מאדעל פראמפטס: "what is machine learning", "explain the water cycle", "the benefits of exercise". מען האט געהאפט ער וועט ענטפערן ריכטיג פון Wikipedia-טעקסט, אבער ער גיט נאר צופעליגע זאצן פון Wikipedia אן שום זינען. דאס איז די צייט צו fine-tune'ן.

• מען אינסטאלירט PEFT און bitsandbytes.

- מען לאדנט Facebook'ס OPT-125M מאדעל – א פראקטישער מאדעל וואס איז קליין גענוג צו טרייַנירן אויף Google Colab.
- LoRA configuration – דער וויכטיגסטער טייל: עס פרירט איין אלע ארגינעלע weights און לייגט צו קליינטשיגע טרייַנירבארע מאטריצעס נאר צו די attention שיכטן. print_trainable_parameters ווייזט אז נאר בערך 0.5% פונעם מאדעל ווערט טאקע געטרייַנירט – די זעלבע רעזולטאטן מיט אסאך ווייניגער פאראמעטערס.
- עס קומט ארויס א טעות אז dataset_text_field איז אן "unexpected keyword"; מען פארריכט עס דורך צולייגן אין די טרייַנירונג-פאראמעטערס =dataset_text_field"text", און די טרייַנירונג לויפט.
- מען לאדנט דעם Dolly דאטא-סעט פון Databricks – א דאטא-סעט פון 15,000 מענטשליך-געשריבענע ביישפילן פון נאכפאלגן אנווייזונגען.
- יעדער פראמפט ווערט סטרוקטורירט אין א ריינעם פארמאט: אנווייזונג, קאנטעקסט, און א גוטער ענטפער – דאס לערנט דעם מאדעל נישט נאר שפראך נאר ווי צו פאלגן אנווייזונגען.
- learning rate, logging steps – training arguments אא"וו.

א משל: דער base מאדעל איז ווי א בעל-קריאה וואס ווייסט אסאך אויף אלעס, אבער מען וויל אז ער זאל ספעציפיש ווערן גוט אין פאלגן אנווייזונגען. אנשטאט אים אלעס צו לערנען, גיט מען אים א קורצע טרייַנירונג – דאס איז LoRA fine-tuning. מאכט עס נאך בעסער: אנשטאט צו דערהיינטיגן יעדע זאך וואס ער ווייסט, לייגט מען צו קליינע נאטיצן אויבנאויף מיט ספעציפישע אינפארמאציע, און נאר די ווערן דערהיינטיגט. דערפאר זענען נאר 0.5% פון די פאראמעטערס טרייַנירבאר. Dolly איז 15,000 ביישפילן פון "דא איז אן אנווייזונג, דא איז דער ריכטיגער ענטפער", אזוי אז דער מאדעל לערנט זיך דעם מוסטער פון זיין נוצליך. דער ציל: אנשטאט די צופעליגע זאצן אויף "machine learning" און "water cycle", זאל דער מאדעל געבן ריכטיגע ענטפערס.

סוף פון ערשטן LLM און אריינפיר אין Transformer (3:48:07)

טעסטן דעם פיין-געטונטן מאדעל (3:48:07)

נאכן טרייַנירן ווייזט דער אויספיר אויך סטאטיסטיק: וויפיל טרייַנינג-ביישפילן (training samples) פער סעקונדע און וויפיל טרייַנינג-טריט (training steps) פער סעקונדע עס האט דורכגעמאכט, מיט א באריכט וואס איז פארגעקומען בשעתן טרייַנירן.

דערנאך ווערט דער קאוד געטעסט. די ענטפערס זענען שוין נאענט צו דעם וואס מען האט ערווארטעט. אויף דער פראגע "וואס איז מאשין-לערנען?" האט דער מאדעל געגעבן א ריכטיגן ענטפער: עס איז א באגריף פון קאמפיוטינג, א מאטעמאטישער מאדעל וואס קען ארויסברענגען אינפארמאציע פון דאטא דורך פארשידענע מעטאדן.

אויף דער פראגע וועגן דעם וואסער-ציקל (water cycle) אין פשוטע ווערטער האט עס נישט געגעבן קיין ענטפער, ווייל דער מאדעל איז נאך נישט גענוג פיין-געטונט. דער קורס ערקלערט: וואס מער מען טוט פיין-טונען א מאדעל, אלץ געניער ענטפערט ער, ווייל ער פארשטייט בעסער וואס צו גענערירן פאר יעדע פראגע. וועט מען טרייַנירט מער, קען מען באקומען א פערפעקטן ענטפער.

אויף דער פראגע "וואס זענען די מעלות פון געניטונג (?exercise" האט עס געגעבן א ריכטיגן ענטפער: עס פארבעסערט די שטימונג, מען פילט זיך מער זיכער און קאנצענטרירט, און עס גיט א הפסקה. דאס איז א ריכטיגער ענטפער אנשטאט צופעליגע זאצן, און דאס איז פונקט וואס פיין-טונען טוט.

א צווייטע סעריע טעסטן מיט פארשידענע סארטן פראמפטס (3:49:30)

דערנאך ווערט געשריבן נאך א קאוד מיט פשוטע פראגעס, כדי צו זען צי דער מאדעל איז טאקע פיין-געטונט. די פראמפטס זענען אויסגעקליבן פון פארשידענע טיפן:

- "די הויפטשטאט פון פראנקרייך": א פשוטע פאקט-פראגע.
- "ווי אזוי ארבעט די אינטערנעט": אן ערקלערונג-זאץ.
- "טיפס פאר לערנען פראגראמירן": ערווארטעט א ליסטע, ווייל מען בעט טיפס.
- "שרייב א מעשה אין איין זאץ": א מעשה דארף זיין קרעאטיוו.
- "וואס איז דער אונטערשייד צווישן GPT און ?BERT": ביידע זענען LLMs.

די רעזולטאטן:

- הויפטשטאט פון פראנקרייך: Paris, ריכטיג.
- אינטערנעט: עס האט געענטפערט אז דאס איז "א פרייע open-source וועב-אפליקאציע", און דאס ווערט אנגענומען אלס ריכטיג.
- דריי טיפס פאר פראגראמירן: דא האט עס נישט געגעבן קיין ריכטיגן ענטפער, נאר זאצן ווי "מען קען לערנען פראגראמירן דורך הערן צום פראגראמירן" און "אויב מען איז נישט באקאנט, קען מען געפינען א גוטן וועג צו לערנען", אנשטאט א ליסטע. דער קורס ווארנט: מען דארף פיין-טונען נאך מער, אריינלייגן מער פרטים אין פראמפט און מער טריינירן. לויט דער אפשאצונג איז דער מאדעל בערך 60% געטריינט, ווייל ער גיט שוין ענטפערס.
- מעשה וועגן א ראבאט: "עס זענען דא דריי סארטן ראבאטן... צוויי סארטן ראבאטן... דער אנדערער ראבאט איז א מענטש". דער מאדעל האט געלייענט וואס עס שטייט אין Wikipedia און גיט עס צוריק אלס ענטפער.
- אונטערשייד צווישן GPT און BERT: "א פראגראם וואס איז געמאכט צו העלפן וויסנשאפטלער פארשטיין די גרונט-פיזיק פון מאטעריע". עס האט איבערגעחזרט דעם זעלבן ענטפער ווי פריער, א סימן אז מען דארף נאך פיין-טונען.

נאך מער פיין-טונען, און צוזאמענפאסונג (3:51:14)

דער קורס ווייזט אז נאך מער פיין-טונען האט אויך נישט אינגאנצן געהאלפן, און טייל ענטפערס האבן זיך פארענדערט:

- הויפטשטאט פון פראנקרייך: יעצט דערציילט עס די באפעלקערונג פון פראנקרייך אנשטאט די הויפטשטאט.
- ווי אזוי ארבעט די אינטערנעט: "די אינטערנעט איז א נעץ", א ריכטיגער ענטפער, וואס ערקלערט אויך וואס איז א server, א client און א וועבזייטל.

- דריי טיפס פאר פראגראמירן: "בוי א פראגראמיר-שפראך, לייען ביכער, שפיל שפילן". דאס איז נישט ריכטיג, ווייל שפילן שפילן העלפט נישט לערנען פראגראמירן.
- מעשה וועגן א ראבאט: "א ראבאט איז געמאכט פון ראבאטן מיטן ציל צו לייזן א פראבלעם", "דער ראבאט ווערט קאנטראלירט דורך א באפעל-אביעקט", "מען קען אים ניצן צו באהאנדלען דעם אביעקט", "ער קוקט אויפן אביעקט און רעאגירט אויף באפעלן". דאס ווערט אנגענומען אלס ריכטיג.
- GPT: "א ווייכווארג וואס מאפירט די פיזישע וועלט". דאס איז ווידער ענליך צום פריערדיגן ענטפער.

די לעקציע: מען דארף ווייטער פיין-טונען, מער אנווייזונגען און מער פראמפטס, כדי דער מאדעל זאל לערנען ריכטיג אידענטיפיצירן באגריפן. למשל, דא האט ער "הויפטשטאט" פארשטאנען אלס "באפעלקערונג" און דערנאך דערויף געענטפערט.

צום סוף גיט מען דעם מאדעל א נאמען און מען היט אים אפ (save). אזוי ענדיגט זיך דאס בויען דעם ערשטן LLM. די צוזאמענפאסונג: דער מאדעל נעמט דעם דאטא-סעט וואס מען האט אים געגעבן (דער wiki דאטא-סעט), רייניגט אים, לייענט אים, און גענערירט ענטפערס אויף די פראמפטס. אזעלכע מאדעלן ווערן געוונט אין אסאך אפליקאציעס אין דער ריכטיגער וועלט, ווי דאקומענט-אנאליז, chatbots און אויפשטעלן APIs.

וואס איז א ?Transformer (3:53:51)

דער קורס הייבט אן א נייע טעמע. מען זאל זיך פארשטעלן א transformer ווי א שטארק פארבעסערטע ווערסיע פון דער "predictive text" אייגנשאפט אויפן טעלעפאן. דער טעלעפאן שלאגט פאר וועמען נאר לויט די לעצטע פאר ווערטער וואס מען האט געטייפט, און די פארשלאגן שאפן נישט שטענדיג זאצן וואס האבן א זין. א transformer, אנדערש, באטראכט דעם גאנצן קאנטעקסט פון דעם וואס מען שרייבט. ער קוקט אויף דער גאנצער מעלדונג, פארשטייט דעם קאנטעקסט און שלאגט פאר ווערטער וואס פאסן צוזאמען. אזוי האלט ער אן אויג אויפן גרעסערן בילד, און דער טעקסט פליסט נאטירלעך און האט א זין.

די טעכנישע דעפיניציע: א transformer-מאדעל איז א זייער קלוגער נויראל-נעץ וואס קען פארוואנדלען איין סארט אינפוט אין אן אנדערן. דער נאמען שטאמט פון א Google ארטיקל פון 2017, וואו פארשער האבן געפונען א וועג צו לערנען א נויראל-נעץ איבערצוצן ענגליש אויף פראנצויזיש זייער גענוי, און אסאך שנעלער ווי אנדערע מעטאדן. עס האט זיך אבער ארויסגעשטעלט אז transformers זענען נוצליך פאר אסאך מער ווי איבערזעצונג: זיי קענען שאפן טעקסט, בילדער, און אפילו אנווייזונגען פאר ראבאטן. זיי זענען שטארק אין פארשטיין באציאונגען צווישן פארשידענע סארטן דאטא, למשל פארוואנדלען גערעדטע אנווייזונגען אין בילדער אדער אין ראבאט-באפעלן.

transformers זענען דער רוקנביין פון די גרויסע שפראך-מאדעלן ווי ChatGPT, Google Search, DALL-E און Microsoft Copilot. כמעט יעדעס מאל וואס מען ניצט אן אפליקאציע וואס פארשטייט מענטשליכע שפראך, ניצט זי מסתמא א transformer.

דער "געהיימער סוד" איז attentionאויפמערקזאמקייט). ווי א מענטש גיט עקסטרע אכטונג אויף געוויסע ווערטער אין א זאץ ווייל זיי העלפן פארשטיין וואס גייט פאר, אזוי ניצט א transformer attention צו קאנצענטרירן זיך אויף די וויכטיגע ווערטער און שטיקלעך דאטא, און אזוי פארשטייט ער בעסער דעם קאנטעקסט.

פארוואס בויט מען טעקסט ווארט נאך ווארט? (3:56:13)

עס קען זיין חידוש אז א transformer בויט טעקסט ווארט נאך ווארט. אזוי שאפן מענטשן נישט קיין זאצן, און אזוי ארבעטן נישט אנדערע מאשין-לערנען מאדעלן. א מענטש הייבט אן מיט א גרונט-געדאנק און פארבעסערט אים אינמיטן, און לייגט צו מער פרטים און פיינקייטן. ענליך, ביי בילד-געניראציע הייבן נויראל-נעץ מאדעלן אן מיט א גראבן ענטוווורף און פארבעסערן אים ביסלעכווייז ביז ער איז פונקט ריכטיג.

transformers נעמען אן אנדערן צוגאנג, און עס איז דא א גוטע סיבה דערפאר: ער ארבעט אויסערגעווענליך גוט. די הויפט-סיבה פאר זייער הצלחה איז אז זיי זענען זייער גוט אין נאכפאלגן דעם קאנטעקסט. זיי פארשטייען דעם אלגעמיינעם באדייט און פלוס פונעם טעקסט, און קענען דערפאר אויסקלייבן דאס בעסטע קומענדיגע ווארט צו פירן דעם געדאנק ווייטער גלאט. כאטש עס זעט אויס אומגעווענליך, ברענגט דער מעטאד שטארקע רעזולטאטן, ווייל זיי כאפן דעם קאנטעקסט אומגעווענליך גוט און דער טעקסט בלייבט צווזאמענהאנגיג.

ווי אזוי ווערט א transformer געטריינט, מיט א ביישפיל (3:57:29)

מען טריינירט מיט אסאך דאטא, טאקע מיט אלע דאטא אויף דער אינטערנעט. ווען מען טייפט "hello, how are you", ווייסט ער לויט דעם טעקסט אויף דער אינטערנעט אז דאס בעסטע קומענדיגע ווארט איז "you". גיט מען א מער קאמפליצירטן באפעל, למשל "write a story", רעכנט ער אויס אז א גוט ערשט ווארט איז "once". דאס ווארט לייגט ער צו צום באפעל, דערנאך רעכנט ער אויס אז דאס קומענדיגע ווארט איז "upon", און אזוי ווייטער, ווארט נאך ווארט, ביז די מעשה איז פארטיג.

אין דער לעקציע ווערט עס געוויזן שריט ביי שריט:

- באפעל: "write a story", ענטפער: "once".
- ווידער "write a story" (מיט דעם וואס איז שוין דא), ענטפער: "once upon".
- דערנאך: "once upon a".
- דערנאך: "once upon a time".

אזוי זעט מען ווי ער בויט ווארט נאך ווארט, און אזוי ארבעטן transformers באמת.

די ארכיטעקטור: Tokenization און Embedding (3:59:47)

די ארכיטעקטור קען אויסזען קאמפליצירט, אבער מען צעטיילט זי אויף אירע הויפט-טיילן:

- Tokenization
- Embedding
- Positional encoding
- Transformer block

Tokenization איז א יסודותדיגער שריט אין natural language processing. מען צעטיילט א שטיק טעקסט, ווי א זאץ אדער א דאקומענט, אויף קלענערע אייגהייטן וואס הייסן tokens. די tokens קענען זיין גאנצע ווערטער, טיילן פון ווערטער (subwords) אדער אפילו איינצלנע אותיות, לויט דער tokenization-סטראטעגיע וואס מען ניצט.

דערנאך דארף מען פארוואנדלען די ווערטער אין נומערן, און דערפאר ניצט מען embedding. אויב צוויי שטיקער טעקסט זענען ענליך, זענען די נומערן אין זייערע וועקטארן ענליך "component-wise", דאס הייסט יעדעס פאר נומערן אויף דער זעלבער פאזיציע איז ענליך. זענען די צוויי שטיקער טעקסט אנדערש, זענען אויך די נומערן אין זייערע וועקטארן אנדערש. אין דעם ביישפיל באקומט "write" א געוויסע מאטריצע פון נומערן, "a" אן אנדערע, און "story" אן אנדערע. אזוי ווערט יעדעס וואָרט אפגעהיט אין וועקטאר-פארמאט.

וען מען האט שוין וועקטארן פאר יעדעס וואָרט אין זאץ, דארף מען זיי צוזאמענשטעלן אין איין וועקטאר צו באארבעטן. דער געווענליכער וועג איז זיי צוזאמענצורעכענען, איינס נאכן אנדערן, פאזיציע ביי פאזיציע. למשל, די וועקטארן (1, 2) און (3, 4) צוזאמענגערעכנט גיבן 1+3 און 2+4, דאס הייסט (4, 6). אין דעם ביישפיל וואלט "write" געהאט א געוויסן וועקטאר און "a" דעם וועקטאר (3, 4), און מען וואלט זיי צוזאמענגערעכנט כדי צו בויען דעם זאץ.

טראנספארמערס, אטענשאן און גרויסע שפראך-מאדעלן (4:02:38)

וואָרט-וועקטארן און פאזיציאנעלע ענקאדירונג (4:02:38)

דער קורס ענדיגט א ביישפיל ווי מען לייגט צוזאמען די וועקטארן פון וועלטער אין א זאץ. מען רעכנט עס אויס באזונדער פאר יעדן טייל פון די וועקטארן: 1 + 3 און 2 + 4. דער רעזולטאט איז 4 און 6. אזוי באקומט מען א מאטריץ פאר דעם גאנצן זאץ. עס ווערט ערקלערט אז אזוי ארבעטן טראנספארמערס אויף א פשוט'ן אופן.

דא קומט אבער א שווערִיגקייט. וען מען לייגט צוזאמען די וועקטארן אין אן אנדערן סדר, באקומט מען דעם זעלבן רעזולטאט. דאס מיינט אז צוויי זעצער מיט די זעלבע וועלטער, נאר אין אן אנדערן סדר, וועלן האבן דעם זעלבן וועקטאר.

כדי צו פאררעכטן דעם פראבלעם, ניצט מען "פאזיציאנעלע ענקאדירונג" (positional encoding). מען לייגט צו צו די וואָרט-וועקטארן א סעט פון פריער-באשטימטע וועקטארן. די עקסטערע וועקטארן טראגן אינפארמאציע וועגן דעם ארט פון יעדן וואָרט אינעם זאץ. אזוי באקומט יעדער זאץ אן אייגענעם וועקטאר, אפילו וען די וועלטער זענען די זעלבע און נאר דער סדר איז אנדערש.

דער ביישפיל אין דער לעקציע איז דער זאץ "write a story". צום וואָרט "write" גיט מען די פאזיציע 1, צו "a" די פאזיציע 2, און צו "story" די פאזיציע 3. אויב מען בייט איבער דעם סדר פון די וועלטער, וועלן די וועקטארן וואס קומען ארויס זיין אנדערש. אזוי ווערט יעדער זאץ באצייכנט אויף אן אייגענעם אופן. דאס איז דער תירוץ אויפן פראבלעם.

אטענשאן: ווי דער מאדעל פארשטייט קאנטעקסט (4:04:48)

דערנאך קומט "אטענשאן" (attention). דאס לייזט א וויכטיגע שווערִיגקייט. עס טרעפט זיך אפט אז דאס זעלבע וואָרט האט פארשידענע באדייטונגען, און דאס פארמישט שפראך-מאדעלן. די embeddings פירן איבער וועלטער אויף וועקטארן אן צו קוקן אויפן קאנטעקסט. אטענשאן איז א שטארקע טעכניק וואס העלפט שפראך-מאדעלן כאפן דעם קאנטעקסט.

דער קורס ברענגט צוויי זעצער: "the bank of the river" און "money in the bank". אין ביידע איז דא דאס וואָרט "bank", אבער מיט פארשידענע באדייטונגען. אינעם ערשטן זאץ מיינט עס דאס לאנד לעבן טייך. אינעם צווייטן מיינט עס א פינאנציעלע אינסטיטוציע, א באנק. דער קאמפיוטער פארשטייט דאס נישט פון זיך אליין, דעריבער דארף מען אריינגעבן קאנטעקסט.

די אנדערע ווערטער אינעם זאץ קענען דא העלפן. אינעם ערשטן זאץ גיבן "the" און "of" נישט קיין סאך קאנטעקסט. אינעם צווייטן זאץ אבער העלפט "money" צו פארשטיין אז "bank" מיינט א פינאנציעלע אינסטיטוציע.

דער עיקר פון אטענשאן איז אז עס ברענגט נענטער צוזאמען די ווערטער פון א זאץ אינעם embedding-שטח. אין "money in the bank" רוקט זיך דאס ווארט "bank" נענטער צו "money". אין "the bank of the river" רוקט זיך "bank" נענטער צו "river". דורך דער ענדערונג כאפט דאס ווארט "bank" אין יעדן זאץ א שטיקל קאנטעקסט פון די ווערטער ארום.

טראנספארמער-מאדעלן ניצן אבער א מער אנטוויקלטע פארעם, "multi-head attention". דא ניצט מען עטליכע embeddings צוגלייך כדי צו פארבעסערן די וועקטארן און געבן א רייכערן קאנטעקסט. multi-head attention העכערט שטארק די עפעקטיווקייט פון שפראך-מאדעלן ביים פראצעסירן און ביים שאפן טעקסט.

די סאפטמעקס-שיכט און אויסקלייבן דאס קומענדיגע ווארט (4:06:49)

א טראנספארמער באשטייט פון עטליכע שיכטן פון טראנספארמער-בלאקן. יעדער בלאק האט אן אטענשאן-שיכט און א feed forward-שיכט. אינאיינעם ארבעט עס ווי א גרויסער נויראל-נעץ וואס זאגט פאראויס דאס קומענדיגע ווארט אינעם זאץ. דער טראנספארמער גיט א שאצונג (score) פאר אלע ווערטער. א העכערע שאצונג מיינט א גרעסערע מעגליכקייט אז דאס ווארט קומט ווייטער.

דער לעצטער שטאפל פונעם טראנספארמער איז א softmax-שיכט. זי פירט איבער די שאצונגען אויף פראבאבילקייטן וואס קומען צוזאמען צו 1. די ווערטער מיט דער העכסטער שאצונג באקומען די העכסטע פראבאבילקייט. פון די פראבאבילקייטן קען מען אויסקלייבן דאס קומענדיגע ווארט, אויך אויף א צופעליגן אופן.

דער ביישפיל אין דער לעקציע:

- "once" האט א פראבאבילקייט פון 0.5
- "somewhere" האט 0.3
- "there" האט 0.2

מען קלייבט אויס דאס ווארט מיט דער העכסטער פראבאבילקייט, און דאס וועלט דער אויסגאנג פונעם טראנספארמער. נאכן שאפן דעם ערשטן טעקסט, ווי "once upon a time there was a", גיט מען אים אלץ ווייטער אריין האלבע זעצער. אזוי, טריט ביי טריט, אנטוויקלט זיך א פולע מעשה.

פאסט-טריינירן (4:08:23)

עס ווערט ערקלערט אז אפילו ווען מען פארשטייט ווי טראנספארמערס ארבעטן, איז נאך פאראן ארבעט. למשל, מען פרעגט: "וואס איז די הויפטשטאט פון אלזשיריע (?Algeria" לכתחילה וואלט דער טראנספארמער גלייך געענטפערט "Algiers" און ווייטער געגאנגען. אבער טראנספארמערס ווערן טריינירט אויף ריזיגע דאטא-סעטן ווי דער אינטערנעט, און דארט קענען זיי טרעפן אומפולשטענדיגע אדער נישט-שייכותדיגע אינפארמאציע.

למשל, נאך דער פראגע ווען דער הויפטשטאט פון אלזשיריע קען דער ווייטערדיגער זאץ זיין נאך א פראגע, ווי "וויפיל באפעלקערונג האט אלזשיריע?" אדער "וואס איז די הויפטשטאט פון ?Burkina Faso". מענטשן

קלערן איבער איידער זיי ענטפערן. א טראנספארמער אבער מאכט פשוט נאך וואס ער האט געלערנט פון זיין טריינירונג-דאטא, אן צו טראכטן צי דער ענטפער פאסט.

דער תירוץ איז "פאסט-טריינירן" (post-training). פונקט ווי מען לערנט א מענטש נייע פעאיגקייטן, קען מען א טראנספארמער ווייטער טריינירן אויף באשטימטע אויפגאבעס. דורכן טריינירן אויף גרויסע דאטא-סעטן פון פראגעס און ענטפערס קען מען פיין-שטימען (fine-tune) זיינע פעאיגקייטן צו געבן ריכטיגע ענטפערס. פאסט-טריינירן איז נישט באגרעניצט נאר צו פראגעס און ענטפערס; עס פארבעסערט אויך די ארבעט אין פארשידענע אנדערע אפליקאציעס.

גרויסע שפראך-מאדעלן: פון RNN ביז טראנספארמער (4:09:25)

גרויסע שפראך-מאדעלן (LLMs) זענען אנטוויקלטע AI-סיסטעמען. זיי זענען געבויט צו פארשטיין, שאפן און באארבעטן מענטשליכע שפראך. זיי ווערן טריינירט אויף ריזיגע דאטא-סעטן און ניצן דיפ-לערנען טעכניקן כדי פאראויסצוזאגן און צו שאפן טעקסט. דעריבער קענען זיי טון פארשידענע אויפגאבעס: איבערזעצן, קיצור מאכן, און אפילו שעפערישע שרייבן.

דער קורס באהאנדלט דערנאך די אנטוויקלונג פון די מאדעלן. מענטשן זענען טעקסט ווי ווערטער, זעצער און פאראגראפן. פאר א קאמפיוטער איז טעקסט נאר א רייע אותיות. אנפאנגס האט מען גענוצט RNNs (recurrent neural networks) כדי מאשינען זאלן פארשטיין טעקסט. די מאדעלן פראצעסירן איין ווארט אויפאמאל, און דערפאר פארגעסן זיי אמאל די פריערדיגע טיילן פון דער רייע.

אין 2017 איז געקומען דער טראנספארמער-מאדעל און האט אינגאנצן איבערגעדרייט דעם פעלד. אנדערש ווי RNNs, ניצן טראנספארמערס אן אטענשאן-מעכאניזם. דערמיט קענען זיי באטראכטן דעם גאנצן זאץ אדער פאראגראף אויפאמאל, אנשטאט איין ווארט נאך דעם אנדערן. דאס פארבעסערט שטארק דאס פארשטיין פונעם קאנטעקסט.

די שטאפלען פון א טראנספארמער מיט א ביישפיל (4:10:58)

א טראנספארמער צעטיילט קודם דעם טעקסט אויף ווערטער אדער שטיקלעך ווערטער (tokens). די tokens ווערן איבערגעפירט אויף נומערן, און דערנאך אויף embeddings וואס כאפן זייער באדייטונג. דער encoder פראצעסירט די embeddings און שאפט א "קאנטעקסט-וועקטאר" וואס שטעלט פאר דעם גאנצן אריינגאנג.

דער ביישפיל-זאץ איז: "as she said this she looked down at her hands".

- Tokenization: דער זאץ ווערט צעטיילט אויף קלענערע שטיקלעך, tokens. אויפן עקראן זעט מען יעדעס ווארט באזונדער אין גענזפיסלעך.
- Encoding: יעדער token ווערט איבערגעפירט אויף א נומער, ווי מען וואלט געגעבן יעדן ווארט אן אייגענעם נומער.
- Embeddings: די נומערן ווערן וועקטארן מיט פיל דימענסיעס וואס כאפן די באדייטונג. למשל, "she" און "her" קענען האבן ענליכע וועקטארן, ווייל זיי קומען אפט פאר אין ענליכע קאנטעקסטן.
- Context vector: דער encoder פראצעסירט אלע וועקטארן צוזאמען כדי צו פארשטיין דעם גאנצן זאץ. ער שאפט א קאנטעקסט-וועקטאר, ווי א קיצור פון דער באדייטונג פונעם גאנצן זאץ.

• Autoregressive generation: מיטן קאנטעקסט-וועקטאר הייבט דער decoder אן שאפן נייעם טעקסט. ער זאגט פאראויס דאס קומענדיגע ווארט לויט דעם קאנטעקסט וואס ער האט געלערנט. למשל, ער קען שאפן דאס ווארט "and" כדי ווייטער צו פירן דעם זאץ.

די ארכיטעקטור: encoder און decoder (4:12:29)

דער קורס ווייזט א דיאגראם פון א טראנספארמער. די ארכיטעקטור באשטייט פון אן encoder און א decoder, יעדער מיט עטליכע שיכטן.

די שיכטן פונעם encoder:

• Input embedding: דער טעקסט ווערט צעטיילט אויף tokens און איבערגעפירט אויף וועקטארן.

• Positional encoding: לייגט צו אינפארמאציע וועגן דעם ארט פון יעדן token אין דער רייע.

• Multi-head attention: לאזט דעם מאדעל זיך קאנצענטרירן אויף פארשידענע טיילן פונעם אריינגאנג אויפאמאל, און אזוי כאפן פארשידענע זייטן דערפון.

• Add & Norm: לייגט צו דעם אריינגאנג פון דער פריערדיגער שיכט און נארמאליזירט דעם רעזולטאט.

• Feed forward: א פולשטענדיג-פארבונדענע נעץ וואס פראצעסירט וואס די אטענשאן האט ארויסגעגעבן.

• Add & Norm נאכאמאל.

די גאנצע סטרוקטור ווערט איבערגעחזרט N מאל; דאס איז וואס "Nx" באצייכנט.

די שיכטן פונעם decoder:

• Output embedding: ענליך צו input embedding, אבער פאר דער ציל-רייע.

• Positional encoding: לייגט צו פאזיציע-אינפארמאציע צו דער ציל-רייע.

• Masked multi-head attention: פראצעסירט די ציל-רייע, אבער פארדעקט (masking) די קומענדיגע tokens, כדי דער מאדעל זאל נישט קוקן פאראויס.

• Add & Norm.

• Multi-head attention: קוקט אויפן אויסגאנג פונעם encoder, כדי דער decoder זאל זיך קאנצענטרירן אויף די שייכותדיגע טיילן פונעם אריינגאנג.

• Add & Norm.

• Feed forward: נאך א פולשטענדיג-פארבונדענע נעץ.

• Add & Norm.

• Linear און Softmax: די לעצטע שיכט, וואס פירט איבער דעם אויסגאנג פונעם decoder אויף פראבאבילקייטן און זאגט פאראויס דעם קומענדיגן token.

דער encoder פראצעסירט די גאנצע אריינגאנג-רייע און שאפט א סעט ענקאדירטע פארשטעלונגען. דער decoder שאפט דעם אויסגאנג טריט נאך טריט. ער ניצט דערביי דעם ענקאדירטן אריינגאנג און די tokens וואס ער האט שוין פריער ארויסגעגעבן.

פארוואס טראנספארמערס קענען פאראויסזאגן טעקסט (4:15:03)

עס ווערן דערמאנט צוויי הויפט-סיבות: (1) דאס איבעריגע (redundancy) אין שפראך, און (2 און entropy) פאראויסזאגבארקייט.

די הויפט-באגריפן:

- Attention mechanism: לאזט דעם מאדעל אפשאצן ווי וויכטיג יעדער token אינעם אריינגאנג איז, און פארבעסערט דאס פארשטיין פונעם קאנטעקסט.
- Positional encoding: העלפט פארשטיין דעם סדר פון די tokens, וואס איז קריטיש פאר דער באדייטונג אין שפראך.
- Multi-head attention: פארבעסערט די פעאיגקייט צו כאפן פארשידענע פארבינדונגען אין די דאטא, דורכן זיך קאנצענטרירן אויף פארשידענע טיילן פון דער רייע צוגלייך.

די ארכיטעקטור לאזט דעם טראנספארמער באהאנדלען לאנגע רייען און קאמפליצירטע שפראך-אויפגאבעס. דערפאר איז זי געווארן א יסוד פון היינטיגע NLP-אפליקאציעס.

וואס איז א LLM, היסטאריע און ביישפילן (4:15:34)

א גרויסער שפראך-מאדעל איז א סארט AI-אלגאריטם. ער ניצט דיפ-לערנען טעכניקן און גרויסע דאטא-סעטן כדי צו פארשטיין, קיצור מאכן, שאפן און פאראויסזאגן נייעם אינהאלט. LLMs זענען א טייל פון generative AI, ספעציעל געמאכט צו שאפן טעקסט.

היסטאריע: שפראך איז וויכטיג פאר קאמוניקאציע, און אין AI דינען שפראך-מאדעלן צו קאמוניצירן און צו שאפן געדאנקען. איינער פון די ערשטע AI שפראך-מאדעלן איז ELIZA , וואס איז ארויס אין 1966. היינטיגע LLMs, ווי GPT-3 און Megatron-Turing NLG 530B, ברייטערן אויס דעם געדאנק מיט ביליאנען פאראמעטערס. דאס לאזט זיי פארשטיין און שאפן טעקסט אסאך בעסער.

ביישפילן:

- GPT-3: ארויסגעגעבן פון OpenAI אין יוני 2020. ער האט 175 ביליאן פאראמעטערס און קען שאפן טעקסט און קאוד פון קורצע פראמפטס. ער איז זייער וועלסאטיל און קען טון א ברייטע רייע שפראך-אויפגאבעס.
- Megatron-Turing NLG 530B: אנטוויקלט פון Nvidia און Microsoft. ער האט 530 ביליאן פאראמעטערס, ער גלאנצט אין לייען-פארשטאנד און אין natural language inference, און ווערט געראכנט פאר איינער פון די שטארקסטע LLMs ביז היינט.

די פעאיגקייטן פון LLMs (4:17:05)

דער קורס הייבט אן אויסרעכענען די פעאיגקייטן פון די מאדעלן:

- Generation: שאפן מעשיות, מארקעטינג-אינהאלט און נאך. למשל, GPT-3 קען שרייבן אן ארטיקל אדער שאפן שעפערישע שריפטן.
- Summarization: צוזאמענקלאפן לעגאלע דאקומענטן, נאטיצן פון זיצונגען, אדער סיי וועלכן לאנגן טעקסט אין א קורצן קיצור. דאס איז באזונדערס נוצליך.

פּאַראַגראַפֿן, סאַרטן און די עפענע מאדעלן (4:17:35)

נאך פּאַראַגראַפֿייטן פון LLMs: איבערזעצן, קלאַסיפיצירן, chatbots (4:17:35)

דער קורס פירט ווייטער די רשימה פון די פּאַראַגראַפֿייטן פון גרויסע שפּראַך-מאדעלן (LLMs). די פריערדיגע פּאַראַגראַפֿייט, וואס האט זיך אנגעשלאסן אינעם אנהייב פון דעם שטיק, איז געווען נוצבאר אין לעגאלע און קארפּאראטיוועס סביבות.

די נעקסטע פּאַראַגראַפֿייט איז איבערזעצן. א מאדעל קען איבערזעצן טעקסט פון איין שפּראַך אין אן אנדערע, און אפילו איבערמאכן טעקסט אין קאוד. דערפּאר זענען LLMs זייער ווערטפּול סיי פּאר גלאבאלע קאמוניקאַציע, סיי פּאר ווייכווארג-אנטוויקלונג.

דערנאך קומט קלאַסיפיצירן. א מאדעל קען אנאליזירן א טעקסט און באשטימען דעם געפיל (sentiment) וואס ליגט דערין, צוטיילן א טעקסט לויט טעמעס, אדער אנכאפן גיפטיגע און באליידיגנדע שפּראַך. דאס העלפט אויסשטערן אינהאלט אויפן אינטערנעט און פּארשטיין וואס קונים זאגן אין זייערע באמערקונגען.

די לעצטע פּאַראַגראַפֿייט אין דער רשימה איז chatbots. מען בויט ווירטועלע געהילפן און פּראַגע-און-ענטפּער סיסטעמען וואס רעדן מיטן באניצער אויף א מענטשלעכן אופן, און געבן קונה-סערוויס אדער אלגעמיינע אינפּארמאַציע.

אנוועַנדונגען אין פּארשידענע אינדוסטריעס (4:18:05)

דערנאך ווערן געוויזן פּראַקטישע אנוועַנדונגען פון LLMs אין פּארשידענע פעלדער:

- געזונטהייט: מען ניצט LLMs צו פּאַראַויסזאגן די סטרוקטור פון פּראָטעינען, אנטדעקן מוסטערן פון קראַנקהייטן, און אנאליזירן מעדיצינישע טעקסטן און פּאַציענטן-רעקארדס. דאס קען ברענגען דורכברוכן אין מעדיצינישער פּארשונג און אין פערזענליך-צוגעפּאסטער מעדיצין.
- קליניקייט-האנדל (retail): מען פּארבעסערט קונה-סערוויס דורך דינאמישע chatbots, וואס ענטפערן אויף פּראַגעס, געבן רעקאמענדאציעס, און מאכן דאס איינקויפן בעסער.
- ווייכווארג-אנטוויקלונג: LLMs העלפן פּראַגראַמירער שרייבן קאוד, געפינען און פּארעכטן טעותים (debugging), און אפילו לערנען ראבאטן אויסצופירן פיזישע ארבעטן.
- פינאנצן: LLMs קענען צוזאמענפּאסן "earnings calls" (די שמועסן וואו פירמעס באריכטן וועגן זייערע פּארדינסטן), צוגרייטן טראנסקריפּטן פון מיטינגס, און אנאליזירן פינאנציעלע דאקומענטן, כדי צו געבן איינזיכטן און שטיצן באשלוסן.
- מארקעטינג: LLMs ארגאניזירן באמערקונגען פון קונים, טיילן פּראדוקטן איין אין סעגמענטן, און שאפן איבערצייגנדיקן רעקלאמע-אינהאלט.

ווי אזוי LLMs לערנען זיך, און ווי מען פּאסט זיי צו (4:19:05)

דער קורס ערקלערט אז LLMs ווערן טריינירט מיט "unsupervised learning", דאס הייסט לערנען אן אויפזיכט. זיי לערנען זיך פון ריזיגע דאטא-סעטן וואס זענען נישט באצייכנט (אן labels). זיי געפינען אליין מוסטערן און פּארבינדונגען אין דער דאטא, און דאס מאכט זיי פּאַראַג צו טון פּארשידענע ארבעטן, למשל שאפן טעקסט, צוזאמענפּאסן און איבערזעצן.

עס ווערן דערמאנט דריי סארטן לערנען ביי LLMs:

• Zero-shot learning: דער מאדעל קען טון אן ארבעט וואס ער איז נישט ספעציעל געווען טריינירט דערויף. דאס איז מעגלעך דאנק דער בריייטער דאטא וואס ער איז געווען טריינירט מיט.

• One-shot learning: דער מאדעל קען זיך לערנען פון איין אינציגן ביישפיל, און דערמיט טון א געוויסע ארבעט בעסער.

• Few-shot learning: דער מאדעל לערנט זיך פון עטליכע ביישפילן, און דאס פארבעסערט זיין פעאיגקייט צו פארשטיין דעם כלל און אויספירן די ארבעט פינקטלעך.

דערנאך וועלן מיר רעדן וועגן דריי טעכניקן צו צופאסן א LLM:

• Prompt tuning: מען פאסט צו די פראמפטס, אדער די אנווייזונגען וואס מען גיט דעם מאדעל, כדי בעסער צו פירן זיינע ענטפערס.

• Fine-tuning: מען טריינירט דעם מאדעל נאכאמאל אויף ספעציפישע דאטא-סעטן, כדי ער זאל זיין פינקטלעכער ביי געוויסע ארבעטן.

• Adapters: מען לייגט צו קלענערע נעצן צום מאדעל. דאס לאזט אים זיך צופאסן צו ספעציפישע ארבעטן, אן דעם וואס מען זאל דארפן נאכאמאל טריינירן דעם גאנצן מאדעל.

דריי סארטן גרויסע שפראך-מאדעלן (4:20:36)

• Encoder-only מאדעלן: די זענען אויסגעצייכנט פאר ארבעטן וואס פאדערן פארשטיין, למשל קלאסיפיצירן און sentiment-אנאליז. דער ביישפיל איז BERT, דאס הייסט Bidirectional Encoder Representations from Transformers.

• Decoder-only מאדעלן: די זענען שטארק אין שאפן טעקסט, ווי למשל שרייבן מעשיות אדער ארטיקלען. דער ביישפיל איז GPT-3, דאס הייסט Generative Pre-trained Transformer 3.

• Encoder-decoder מאדעלן: די פאראייניגן פארשטיין און שאפן, און דערפאר פאסן זיי גוט פאר איבערזעצן און צוזאמענפאסן. דער ביישפיל איז T5, דאס הייסט Text-to-Text Transfer Transformer.

מעלות און שוועריגקייטן (4:21:07)

מעלות:

• ברייטערן (extensibility): א LLM דינט ווי א יסוד פאר פארשידענע צוגעפאסטע באניץ, און מען קען אים צופאסן צו א סך ארבעטן.

• בייגזאמקייט: מען קען אים ניצן ביי פארשידענע ארבעטן און אינסטאלאציעס, פון chatbots ביז שאפן אינהאלט.

• לייסטונג: ער גיט שנעלע ענטפערס מיט ווייניג פארהאלטונג (low latency).

• פינקטלעכקייט: וואס מער פאראמעטערס און וואס גרעסער דער טריינינג-דאטא-סעט, אלץ פינקטלעכער ווערט ער.

• גרינגקייט פון טריינירן: לויטן קורס ניצט ער באצייכנטע דאטא, און דאס איילט אן דעם טריינירן.

• עפעקטיווקייט: ער פירט אויטאמאטיש אויס רוטינע-ארבעטן, און שפארט דערמיט צייט און מיטלען.

שוועריגקייטן:

• קאסטן פון אנטוויקלונג: מען דארף איינווסטירן א סך אין הארדווער און אין גרויסע דאטא-סעטן.

- קאסטן פון באטרייב: עס קאסט טייער כסדר צו לויפן און אנהאלטן א LLM.
- פארנייגונג (bias): ער קען ירשענען פארנייגונגען וואס געפינען זיך אין דער טריינינג-דאטא.
- עטישע זארגן: עס זענען דא פראגעס ארום פריוואטקייט פון דאטא און ארום שאפן שעדליכן אינהאלט.
- ערקלערבארקייט: עס איז שווער צו ערקלערן ווי אזוי דער מאדעל קומט צו זיינע רעזולטאטן.
- "hallucination": עס איז דא א ריזיקע אז ער וועט געבן אומריכטיגע אדער אומזיניגע ענטפערס.

וועגן דער צוקונפט זאגט דער קורס אז זי איז זייער פארשפרעכנד: די פינקטלעכקייט, די עפעקטיווקייט און די אנוועגדונגען וועלן כסדר בעסער, און ביזנעסער וועלן אלץ מער ניצן LLMs, וואס וועט פארגרעסערן פראדוקטיווקייט און חידושים. דער סך-הכל: LLMs טוישן איבער די וועלט פון AI דורכן פארשטיין און שאפן טעקסט ווי א מענטש. כאטש עס זענען דא שווערigkeiten, איז זייער פאטענציאל איבערצודרייען אינדוסטריעס ריזיג.

משל פון צוויי רעסטאראנטן: פארמאכטע קעגן עפענע מאדעלן (4:22:42)

דא הייבט אן א נייע לעקציע, וואס ערקלערט מיט א משל. שטעלט אייך פאר צוויי רעסטאראנטן פון וועלט-קלאס אויף דער זעלבער גאס.

רעסטאראנט A איז "די געהיימע קיך". דאס עסן איז אויסגעצייכנט, אבער מען צאלט פאר יעדע מאלצייט. די רעצעפטן זענען פארשלאסן, מען קען עס נישט אליין קאכן און נישט טוישן. אויב דער בעל-הבית הייבט מארגן אויף די פרייזן, דארף מען אדער צאלן אדער אוועקגיין. דאס זענען ChatGPT, Claude און Gemini, וואס מען רופט פראפרייטערי (proprietary) אדער "closed source" מאדעלן. מען צאלט פאר יעדן API-רוף, פאר יעדן טאקען, פאר יעדע פראגע, און די פירמע קאנטראלירט אלעס: דעם קאוד, די "weights" און די טריינינג-דאטא.

רעסטאראנט B איז "די עפענע קיך". די זעלבע מדרגה פון טאלאנט, אבער דער קאך פארעפנטלעכט דעם גאנצן רעצעפט אומזיסט אויפן אינטערנעט. מען קען עס קאכן אינדערהיים, מאכן שארפער, צולייגן אן אייגענעם שטיקל, און אויב מען האט עס פארבעסערט, עס צוריק טיילן מיט אנדערע. דאס איז אן open-source LLM: דער קאוד, די weights, און אמאל אויך די טריינינג-דאטא, זענען עפנטלעך. מען קען עס אראפלאדן, שטודירן, טוישן און בויען דערמיט.

אין 2023 איז רעסטאראנט A קלאר געווען בעסער, דערויף זענען אלע מסכים געווען. ביז 2026 איז רעסטאראנט B אזוי גוט געווארן אז קריטיקער קענען קוים מערקן דעם חילוק. און ביי געוויסע "מאכלים", ווי קאוד, ארבעט מיט פיל שפראכן און טעכנישע לאגיק, געווינט טאקע רעסטאראנט B.

וואס איז א LLM: פון אנהייב אן (4:24:14)

דער קורס ערקלערט עס דורך פארגלייכן מיט ווי אזוי א קינד לערנט זיך שרייבן. ער לערנט נישט אויסנווייניג גראמאטיק-כללים פונעם ערשטן טאג. ער לייענט טויזנטער מעשיות, הערט טויזנטער שמועסן, און מיט דער צייט כאפט זיין מוח אויף מוסטערן. ער לערנט זיך וועלכע ווערטער קומען געווענליך איינס נאכן אנדערן. ער לערנט זיך אויך אז "I went to the bank" מיינט עפעס אנדערש לויטן קאנטעקסט: רעדט מען פון געלט (א באנק), אדער פון א טייך (דער ברעג)?

א LLM טוט פונקט דאס זעלבע, נאר ער לייענט ביליאנען בלעטער: Wikipedia, ספרים, וויסנשאפטלעכע ארטיקלען, קאוד, נייעס, סאציאלע מעדיע, אלעס. ער לערנט די מוסטערן אזוי טיף אז ער קען מיט גרויס פינקטלעכקייט פאראויסזאגן וועלכעס וואָרט קומט נאכדעם.

דער "כישוף" ליגט אין דעם: וועו מען שטעלט צוזאם די פאראויסזאגונג פון וואָרט נאך וואָרט טויזנטער מאל, און מען שטעלט אים פראגעס, זעט דער רעזולטאט אויס ווי טראכטן. ער ענטפערט פראגעס, שרייבט קאוד, זעצט איבער שפראכן, און פאסט צוזאם באריכטן פון 100 בלאט. דאס וואָרט "large" (גרויס) מיינט אז ער איז געווען טריינירט אויף ריזיגע מאסן דאטא, און ער האט ביליאנען ביז טריליאנען "פאראמעטערס". "Language model" מיינט אז ער איז ספעציעל געבויט צו פארשטיין און שאפן מענטשלעכע שפראך.

טאקענס און דער קאנטעקסט-פענצטער (4:25:47)

דער ערשטער באגריף איז טאקענס, די גרונט-בויסטיינער. מאדעלן לייענען נישט וועָרטער, זיי ניצן טאקענס. א טאקען איז בערך א שטיקל וואָרט: דאס וואָרט "playing" קען צעטיילט ווערן אין "play" און "ing". א געווענליכער זאץ איז בערך 15 ביז 20 טאקענס.

פארוואס איז דאס וויכטיג? ווייל מאדעלן האבן א גרענעץ ויפיל טאקענס זיי קענען באהאנדלען אויף איינמאל. דאס רופט זיך דער "context window" (קאנטעקסט-פענצטער), וואס מען קען זיך פארשטעלן ווי דעם מאדעל'ס ארבעטס-זכרון. אלטע מאדעלן האבן געהאט 4,000 טאקענס, בערך דריי בלאט טעקסט. היינט האט Llama 4 א קאנטעקסט-פענצטער פון 10 מיליאן טאקענס. מען קען אים פיטערן א גאנצע ביבליאטעק, און ער וועט זיך אלעס געדענקען אין איין שמועס.

פאראמעטערס און טריינירן (4:26:18)

פאראמעטערס זענען דעם מאדעל'ס זכרון. זיי זענען די נומערן וואס באשטימען ווי אזוי דער מאדעל טראכט. שטעלט אייך פאר ביליאנען קליינע קנעפלעך אויף א "mixing board" (א סאונד-פולט). יעדער קנעפל קאנטראלירט א קליינעם צד פון ווי שפראך ווערט פארשטאנען און געשאפן. מער קנעפלעך מיינט מער פיינקייט און מער אינטעליגענץ, אבער אויך מער וואג צו טראגן. א מאדעל מיט 7 ביליאן פאראמעטערס האט 7 ביליאן קנעפלעך. DeepSeek V3 האט 685 ביליאן אינגאנצן. דער קלוגער פונקט איז אז מען דארף נישט אלע אקטיוו אויף איינמאל. דאס וועט ערקלערט ווערן ווייטער אינעם אפטייל וועגן "mixture of experts".

טריינירן מיינט לערנען דעם מאדעל. מען פאסט צו אלע פאראמעטערס ווידער און ווידער, און מען ווייזט דעם מאדעל ביליאנען ביישפילן, ביז ער לערנט זיך פינקטלעך פאראויסזאגן טעקסט. עס איז ווי א תלמיד וואס טוט צענדליגע מיליאנען איבונג-שאלות. די רעכן-קראפט וואס מען דארף איז ריזיג. דערפאר קאסט טריינירן א "frontier" מאדעל (פון די מערסט פארגעשריטענע) הונדערטער מיליאנען דאלער, און עס נעמט חדשים אויף טויזנטער GPUs. די שיינקייט פון open-source איז אבער אז איינער אנדערש האט שוין געטון די טייערע ארבעט. מען לאדט נאר אראפ דעם רעזולטאט, און מען שפרינגט גאנץ איבער דעם טריינינג-חשבון.

Base מאדעל קענען fine-tuned מאדעל (4:27:52)

דער קורס באטאנט אז דאס איז א פונקט וואס רוב מענטשן ערקלערן נישט, כאטש עס מאכט א גרויסן חילוק אין דער פראקטיק. Open-source LLMs קומען אין צוויי סארטן:

- Base מאדעל, "דער רויער מוח": דאס איז דער רויער רעזולטאט פון טריינירן אויף ביליאנען בלעטער. ער קען זייער גוט שפראך-מוסטערן, אבער ער וויסט נישט דווקא ווי צו פאלגן אייערע אנווייזונגען. פרעגט מען

אים "וואס איז די הויפטשטאט פון פראנקרייך?", קען ער פשוט ממשיך זיין צו שרייבן טעקסט אין א Wikipedia-סטיל, אנשטאט גלייך צו ענטפערן "Paris". Base מאדעלן זענען פאר פארשער וואס ווילן בויען אויף זיי. זיי זענען דער יסוד, נישט דאס פארטיגע פראדוקט.

- Fine-tuned מאדעל, "דער נוצלעכער מוח": מען נעמט א base מאדעל און מען טריינירט אים ווייטער אויף ספעציפישע ביישפילן: פארן פון פראגעס מיט ענטפערס, אנווייזונגען, און אינהאלט פון א געוויסן פאך. דאס לערנט אים טאקע צו פאלגן אנווייזונגען, גלייך צו ענטפערן, און זיך פירן אויף א הילפסבערן אופן. וועו מען ניצט ChatGPT, ניצט מען א fine-tuned מאדעל: OpenAI האט געמומען זייער base מאדעל און אים fine-tuned צו זיין א געהילף. כמעט יעדע פראקטישע אפליקאציע ניצט fine-tuned מאדעלן, נישט רויע base מאדעלן.

Continuous pre-training קעגן fine-tuning (4:29:24)

עס איז דא אויך "continuous pre-training", וואס איז אנדערש פון fine-tuning. Continuous pre-training מיינט פיטערן דעם מאדעל נייעם טעקסט פון א געוויסן פאך, כדי צו פארבעסערן זיין וויסן, למשל צולייגן אלע מעדיצינישע זשורנאלן וואס זענען ארויס אין די לעצטע 2 יאר. Fine-tuning לערנט אים דערנאך ווי צו ניצן דאס וויסן אויף א גאר ספעציפישן אופן. ביידע טוישן די weights פונעם מאדעל, און ביידע פאדערן רעכן-קראפט, אבער זיי דינען פארשידענע צוועקן.

דער עיקר-לימוד: כמעט פאר יעדן אמת'דיגן באניץ וויל מען א fine-tuned מאדעל, נישט קיין base מאדעל. וועו מען זוכט אויף Hugging Face, זאל מען קוקן נאך וועריאנטן וואס הייסן "instruct" אדער "chat". די זענען fine-tuned צו פאלגן אנווייזונגען.

open-source קעגן פראפרייטערי: וואס עס ליגט אויפן שפיל (4:29:56)

ערשטער סצענאר, "די שפיטאל-פראבלעם": מען פירט א שפיטאל און מען וויל אז AI זאל העלפן די דאקטוירים צוזאמענפאסן די היסטאריע פון פאציענטן און אנכאפן געפערלעכע צוזאמענשטויסן פון מעדיצינען. ChatGPT איז אויסגעצייכנט, אבער די לעגאלע מאנשאפט הייבט אויף א האנט: יעדעס מאל וואס א דאקטאר שרייבט א פאציענט-נאטיץ אין ChatGPT, גייט די דאטא צו OpenAI 'ס סערווערס. דאס איז א פארלעצונג פון HIPAA, און פריוואטקייט-געזעצן פאר פאציענטן כמעט אין יעדן לאנד פארבאטן דאס. מיט אן open-source LLM אינסטאלירט מען אים אויף די אייגענע סערווערס פונעם שפיטאל, און די דאטא פארלאזט קיינמאל נישט דעם נעץ. נול לעגאלע ריזיקע, פולע צוהאלטונג צום געזעץ, און לויטן קורס טוען דאס טאקע געזונטהייט-ארגאניזאציעס יעצט.

צווייטער סצענאר, "דאס איינשליסן פון א סטארט-אפ": מען בויט א פראדוקט אויף OpenAI'ס API, און דער ביזנעס וואקסט. דערנאך טוישט OpenAI די פרייזן, און די קאסטן פאר יעדע פראגע ווערן דריי מאל אזוי הויך. אדער זיי דערהיינטיגן דעם מאדעל, און דער פראדוקט צעברעכט זיך אויף סובטילע אופנים. מען האט קיין שום קראפט צו פארהאנדלען, מען איז אינגאנצן אפהענגיג. אן "open weight" מאדעל גיט עפעס וואס האט נישט קיין פרייז: קאנטראל איבער דעם אייגענעם AI-סיסטעם. מען קלייבט אויס דעם מאדעל, מען באשטימט וואו ער לויפט, און מען פירט די אייגענע דאטא. קיינער קען דאס נישט צונעמען, און קיינער קען נישט טוישן די פרייזן איבער נאכט. דערנאך גייט דער קורס איבער צו די לייסענסעס.

אפן-סאורס מאדעלן: לייסענסעס, פראיעקטן און די שפּיץ־מאדעלן (4:31:34)

לייסענסעס: "אפן-סאורס" מיינט נישט אלעמאל פרייהייט (4:31:34)

דער קורס הייבט אן מיט א ווארענונג וואס כאפט אסאך מענטשן אומגעווארנט: "אפן-סאורס" מיינט נישט אלעמאל אז מען מעג טון וואס מען וויל. די לייסענסעס זענען זייער פארשידן.

- Apache 2.0 מיינט אינגאנצן פרייע קאמערציעלע באנוץ, אן שום באגרעניצונגען.
- Meta Lama 4 לייסענס ערלויבט קאמערציעלע באנוץ, אבער נאר ווען מען האט ווייניגער ווי 700 מיליאן אקטיוועע באניצער א חודש. דאס האלט אפ Microsoft און OpenAI פון בויען קאנקורירנדע פראדוקטן אויף Meta'ס ארבעט. א קליינער סטארט-אפ איז אינגאנצן באוואָרנט, אבער מען דארף לייענען דעם לייסענס איידער מען שיקט ארויס א פראדוקט.
- DeepSeek קומט ארויס אונטער MIT, אן שום פליכטן פאר די וואס באנוצן עס ווייטער.
- Qwen 3 איז Apache 2.0 אין אלע גרייסן.
- Gemma 4 איז אויך Apache 2.0.

עס ווערט באטאנט אז די אמת'ע פרייע לייסענסעס זענען א גרויסע זאך פאר די וואס בויען קאמערציעלע פראדוקטן.

די מארק-ווירקליכקייט: מער ווי האלב לויפט אויף אייגענע סערווערס (4:32:20)

דער קורס ברענגט א צאל וואס, ווי עס ווערט געזאגט, וועט איבערראשן יעדן וואס מיינט אז ChatGPT באהערשט די AI וועלט: מער ווי 50% פונעם מארק פון גרויסע שפראך-מאדעלן לויפט "on premises", ד.ה. אויף די אייגענע סערווערס פון די פירמעס, און נישט דורך קלאוד API'ס.

ChatGPT באקומט די גאנצע פרעסע מיט אירע 180 מיליאן באניצער, אבער שטיל אין הינטערגרונט פאסירט מער ווי האלב פון דער באנוץ פון שפראך-מאדעלן אין ארגאניזאציעס אויף מאדעלן וואס זיי פארמאגן און קאנטראלירן אליין, אנשטאט צו דינגען צוטריט ביי א פארקויפער. עס ווערט אונטערגעשטראכן אז דאס איז נישט א נישע פאר פראגראמירער, נאר הויפטשטראם-באנוץ פון אפן-סאורס AI ביי גרויסע פירמעס.

וועגן דעם טעמפא: זינט אנהייב 2023 האבן זיך די ארויסגעגעבענע אפן-סאורס מאדעלן כמעט פארטאפלט אין פארגלייך מיט פארמאכטע מאדעלן, בשעת די פארקויפער האבן געטוישט זייערע פרייז-סטרוקטורן ווי עס איז זיי איינגעפאלן. יעדן חודש קומען ארויס נייע מאדעלן, נייע "fine-tunes" און נייע ארכיטעקטור-דורכברוכן, און די קאמיוניטי באוועגט זיך שנעלער ווי די גרויסע לאבארעטאריעס קענען כאפן. דער קורס שליסט: דאס איז נישט די צוקונפט פון AI, נאר די געגענווארט.

אכט סארטן פראיעקטן וואס מען קען אליין בויען (4:33:35)

דער קורס בעט דעם לייענער צו טראכטן "וואס קען איך פערזענליך בויען", ווייל אפן-סאורס שפראך-מאדעלן זענען טאקע צוגענגליך פאר יחידים, קליינע גרופעס און סטודענטן, נישט נאר פאר פארשער און גרויסע פירמעס.

- טעקסט-געניראציע אפליקאציע: שרייב-געהילפן, אימעיל-שרייבער, בלאג-געניראטארן, שעפערישע מעשה-געצייג, אלעס וואס שאפט א פאלירטן טעקסט. אז מען טריינירט עס ווייטער (fine-tune) אויף דעם

קול פונעם בראנד אדער אויף א געוויסן געביט, האט מען א פראדוקט צו פארקויפן, למשל א לעגאלער שרייב-געהילף וואס רעדט ווי אן אדוואקאט. דאס זענען אמת'ע פראדוקטן וואס ווערן יעצט געבויט.

• קאוד-געניראציע און איבערקוקן: מאדעלן געטרעינירט אויף GitHub קאוד, ווי StarCoder וואס דעקט איבער 600 פראגראמיר-שפראכן, קענען שרייבן, דיבאגן, איבערקוקן און ערקלערן קאוד. דאס גייט ווייט העכער ווי אויטאמאטישע פארענדיגונג: מאדערנע קאוד-מאדעלן פארשטייען א גאנצע קאוד-באזע, שלאגן פאר ארכיטעקטור-פארבעסערונגען, געפינען זיכערהייט-שוואכקייטן און שרייבן טעסט-פאלן.

• AI טשעטבאטס: א טשעטבאט וואס קען דיין געשעפט, פראדוקטן, פאליסיס און טאן, געטרעינירט אויף דער אייגענער פירמע-דאטא און לויפט אויף די אייגענע סערווערס. נישט קיין אלגעמיינער "GPT wrapper", נאר אן אמת'ער בראנדירטער AI געהילף וואס האלט די דאטא אינעווייניג.

• צוזאמענפאסן אינהאלט: מען גיט אריין א באריכט פון 50 זייטן און באקומט א צוזאמענפאסונג פון דריי פאראגראפן מיט די הויפט-שריט צו טון, פאר זשורנאליזם, לעגאלע, פינאנץ און פארשונג גרופעס. נאר דאס אליין, ווערט געואגט, איז ווערט די קאסטן, און עס שפארט שהען יעדן טאג.

• שפראך-איבערזעצונג: נישט קיין אלגעמיינע איבערזעצונג, נאר געשליפן פאר ספעציפישע ווערטער-אוצרות: מעדיצינישע איבערזעצונג וואס טרעפט די טערמינאלאגיע, לעגאלע וואס היט אפ דעם געניעם טייטש, טעכנישע הענטביכער איבערגעזעצט געמוי אויף 30 שפראכן. אלגעמיינע געצייג מאכן אין דעם כסדר טעותים, געשליפענע אפן-סאורס מאדעלן טרעפן עס.

• ווירטועלע לערנונג (tutoring): אן אפליקאציע וואס פאסט זיך צו צו ווי א סטודענט לערנט, פרעגט פראגעס לויט די שוואכע פלעקן און ערקלערט א באגריף אויף 10 פארשידענע אופנים ביז עס כאפט זיך. אלעס פערזענליך, און ווייל עס לויפט אויף דער אייגענער אינפראסטרוקטור, פארלאזט די סטודענט-דאטא קיינמאל די פלאטפארמע.

• געפיל-אנאליז און בראנד-אויפפאסונג: באארבעטן טויזנטער קונה-רעצענזיעס, הילף-טיקעטס און סאשעל-מידיע פאסטס, און געבן די פראדוקט-גרופעס אינפארמאציע אין רעאלע צייט: זענען די קונים צופריידן? וואס באקלאגן זיי זיך? וועלכע פיטשערס בעטן זיי כסדר? אויטאמאטישע געפיל-אנאליז אין גרויסן מאס.

• אינהאלט-מאדעראציע: פאר פלאטפארמעס מיט אינהאלט פון באניצער, פארומס, גיים-מארקעטפלעיסעס, קענען געשליפענע שפראך-מאדעלן אויסגעפינען שעדליכן, באליידיגנדן אדער אומפאסיגן אינהאלט אין גרויסן מאס, פריוואט אויף די אייגענע סערווערס, מיט פולע קאנטראל איבער די מאדעראציע-כללים.

וועך באנוצט זיי טאקע (4:36:30)

• NASA און IBM: זיי האבן זיך צוזאמענגעשלאסן צו בויען אן אפן-סאורס געאגראפישן יסוד-מאדעל (foundation model) מיטן נאמען Prithvi, געטרעינירט אויף סאטעליט-בילדער און ערד-אבזערוואציע דאטא. דאס איז נישט קיין געווענליכער טעקסט-מאדעל, נאר א ספעציאליזירטער מאדעל, פריי צו באקומען אויף Hugging Face, וואס העלפט וויסנשאפטלער נאכפאלגן אויסהאקן וועלדער, פאראויסזאגן מבול-געגנטן און אנאליזירן קלימאט-דאטא אין גרויסן מאס. א ביישפיל ווי אפענע יסוד-מאדעלן עפענען ספעציפישע וויסנשאפט וואס וואלט געדויערט יארן צו בויען פון אנהייב.

• געזונטהייט: שפיטעלער איבער דער וועלט באנוצן געשליפענע אפן-סאורס מאדעלן פאר דיאגנאז-הילף, באהאנדלונג-אפטימיזאציע און קליניש דאקומענטירן, אלעס אויף זייער אייגענער פריוואטער אינפראסטרוקטור, אינגאנצן לויט HIPAA. די פאציענט-דאטא פארלאזט נישט דעם בנין.

- פינאנץ: FinGPT, ספעציעל געשליפן פאר דער פינאנץ-אינדוסטריע. עס אנאליזירט פארדינסט-באריכטן, פאסט צווזאם פינאנציעלע נייעס, אפשאצט מארק-געפילן און העלפט מיט ריזיקע-אנאליז. באנקען און העדזש-פאנדס לויפן עס פריוואט, און די סענסיטיווע דאטא בלייבט אינעווייניג.
- זשורנאליזם און פארלאגן: ארגאניזאציעס באנוצן אינעווייניג אפן-סאורס מאדעלן צו אנאליזירן און צווזאמענפאסן גרויסע מאסן אינפארמאציע, אויסגעפינען הויפט-מעשיות און נאכפאלגן מוסטערן איבער טויזנטער ארטיקלען, אן שיקן די אייגענע זשורנאליסטישע דאטא ארויס פון דער רעדאקציע. וועז מען פארשט א מעשה, ווייל מען נישט אז די זוך-פראגעס זאלן גיין צו א דריטער פארטיי.

צען שפיץ-מאדעלן: #DeepSeek V3.2 1 (4:38:17)

דער קורס רופט דאס דעם וויכטיגסטן חלק: צען מאדעלן, מיט וואס מאכט יעדן ספעציעל, פאר וועמען ער איז, און די אמת'ע ספעציפיקאציעס.

DeepSeek V3.2 ווערט אנגערופן "דער דיסראפטער וואס האט צעבראכן Wall Street ". די ארכיטעקטור איז Mixture of Experts, 685 ביליאן פאראמעטערס אינגאנצן, אבער נאר 37 ביליאן אקטיוו פאר יעדן טאקען.

- DeepSeek Sparse Attention (DSA) שניידט די קאמפלעקסקייט פון attention פון קוואדראטיש צו כמעט לינעאר, א ריזיגע זאך פאר לאנגן קאנטעקסט.
- Multi-head Latent Attention דריקט צווזאם דעם KV cache אן שאטן דער קוואליטעט.
- רעזאנירן איז איינגעבויט גלייך אין באנוץ פון געצייג, מען קען בויען אגענטן וואס טראכטן איידער זיי טוען.
- עס האט געוואונען גאלד ביי IMO 2025 און IOI 2025.
- קאנטעקסט: 128,000 טאקענס. לייסענס: MIT פאר דעם קאוד, DeepSeek לייסענס פאר די וואגן (weights), פריי פאר קאמערציעלע באנוץ אונטער 1 מיליאן איינקונפט.
- API פרייז: בערך 0.42 פאר א מיליאן טאקענס, א בראכטייל פון די פארמאכטע אלטערנאטיוון.

עס איז דאס בעסטע פאר קאודירן, קאמפליצירטע רעזאנירונג, מאטעמאטיק, אגענט-ארבעטספלאוס און פראדוקציע ווו מען היט אויף די קאסטן. דער קורס ערקלערט: DeepSeek האט אנגעהויבן דעם "ערדציטערניש", אבער אויסער דעם דראמע איז עס טעכניש אויסערגעווענליך. די sparse attention מיינט אז מען קען לויפן א פראמפט פון 128,000 טאקענס אן דאס GPU זכרון זאל אויפרייסן. די צוויי ווערסיעס פון V3.2 געבן א ברירה: די "Thinking" ווערסיע בויט אגענטן וואס רעזאנירן איידער זיי טוען; די "Speciale" ווערסיע נעמט אוועק די געצייג און דרייט ארויף דאס רעזאנירן צום מאקסימום, און זי האט געוואונען גאלד-מעדאלן אין מאטעמאטיק און קאודירן אלימפיאדעס. MIT לייסענס פארן קאוד מיינט מאקסימום קאמערציעלע פרייהייט.

#Meta Llama 4 2 (4:40:13)

דער יסוד פונעם אפן-סאורס סיסטעם. צוויי ווערסיעס צום אראפלאדן:

- Scout: 17 ביליאן אקטיוו, 109 ביליאן אינגאנצן, 16 עקספערטן. עס פאסט אויף איין H100 וועז מען קוואנטיזירט עס אויף INT4, פאר ערנסטע ארבעט אן ריזיגע הארדווער. קאנטעקסט פון 10 מיליאן טאקענס, אויסערגעווענליך פאר לאנגע דאקומענטן.
- Maverick: 17 ביליאן אקטיוו, 400 ביליאן אינגאנצן. דאס באנוצט Meta אינעווייניג פאר WhatsApp, Messenger און Instagram, און אויף בענטשמארקס שלאגט עס GPT-4o.

עס איז פון נאטור מולטימאדאל: טעקסט און בילדער אריין, טעקסט ארויס; געטריינירט אויף 200 שפראכן, fine-tuning געשטיצט פאר 12. זיכערהייט איז איינגעבויט: Llama Guard פילטערט אריינגאנג און ארויסגאנג, Prompt Guard כאפט "jailbreak" און "injection" פרואוון, CyberSecEval אפשאצט זיכערהייט-שוואכקייטן. לייסענס: Meta Llama 4 Community License, קאמערציעל ערלויבט אונטער 700 מיליאן אקטיוו באניצער א חודש. בעסט פאר אלגעמיינע קאמערציעלע פראדוקטן, מולטימאדאלע אפליקאציעס אויסער דעם EU, און פירמע-fine-tuning. דער קורס זאגט אז Scout איז דער פראקטישער טעגלעכער ארבעטס-פערד, Maverick איז וואס Meta באנוצט פאר אירע אייגענע פראדוקטן, און די זיכערהייט-געצייג זענען ערנסט. א ווארענונג פאר די וואס בויען אין אייראפע: די זען-פעאיגקייטן זענען בלאקירט אין EU לויט Meta'ס אייגענער פאליסי, מען דארף דאס נאכקוקן איידער מען פלאנט א פראדוקט ארום פארשטיין בילדער.

#AI Mistral 3 (4:41:51)

דער עפעקטיווער פראנצויזישער סטארט-אפ וואס שלאגט ווייט העכער ווי זיין גרייס. די Ministral מאדעלן פון 3 ביליאן און 8 ביליאן לויפן אויף טעלעפאנען, מיט ענטפער-צייט אונטער 500 מיליסעקונדעס אויף נייע מאביל-הארדוועיר, און זיי שלאגן די גלייכע מאדעלן פון Google און Microsoft אויף רוב בענטשמארקס. נאטירליכע "function calling": עס פארשטייט באנוץ פון געצייג אן ספעציעלע פראמפט-אינזשעניריע. קאנטעקסט בערך 128,000 טאקענס ביי די גרעסערע מאדעלן, גענוג פאר רוב פירמע-דאקומענטן. Mixture of Experts אקטיווירט נאר די נויטיגע טיילן און שניידט קאסטן אן שאטן קוואליטעט. לייסענס איז צעטיילט: Apache 2.0 פאר די קליינע מאדעלן, קאמערציעלע לייסענס פאר די גרעסערע. עס איז GDPR קאנפארם, פאפולער ביי אייראפעישע ארגאניזאציעס צוליב דאטא-ווואוינארט. בעסט פאר מאביל AI, אפליין באנוץ, אייראפעישע רעגולאציע און אגענט-ארבעט וואס דארף געצייג. Mistral איז געקומען פון ערגעץ, א קליינע גרופע געווען DeepMind און Meta פארשער אין פאריז. די קליינע מאדעלן לויפן טאקע אויף טעלעפאנען, נישט נאר טעאָרעטיש, און "function calling" איז א גרעסערע זאך ווי מען מיינט: אנדערע מאדעלן דארפן זייטן פון פראמפטס כדי צו רופן געצייג פארלעסלעך, און Mistral טוט עס פשוט ווייל עס איז פון אנהייב אזוי געבויט.

#Google Gemma 3 / 4 Gemma 4 (4:43:22)

Google'ס אפן-סאורס "מתנה", פינף גרייסן פון קליינע געצייג ביז ארבעטס-סטאנציעס: 270 מיליאן, 1 ביליאן, 4 ביליאן, 12 ביליאן, 27 ביליאן. דער 27B מאדעל שלאגט Llama 405B, DeepSeek V3 און o3-mini אויף LMArena . דער 270 מיליאן מאדעל באנוצט נאר 0.75% באטעריע פאר 25 שמועסן אויף א Pixel 9 Pro. 4B און העכער זענען מולטימאדאל (טעקסט און בילדער), 1B נאר טעקסט. קאנטעקסט 128,000 טאקענס, ארויף פון 8,000 אין דער פריערדיגער ווערסיע. איבער 140 שפראכן, אייגנעבויטע "function calling" פאר אגענטן. ShieldGemma 2 איז א באגלייט-מאדעל פאר זיכערהייט וואס קלאסיפיצירט שעדליכן אינהאלט לויט אייגענע כללים און שלאגט GPT-4o mini. עס ארבעט מיט Keras, JAX, PyTorch, Hugging Face און vLLM. בעסט פאר עדוש-געצייג, מאביל, פילשפראכיקייט און פאראנטווארטליכע באנוץ. א 27B מאדעל וואס שלאגט עפעס 15 מאל גרעסער איז נישט קיין טריק, נאר distillation טריינירן און קלוגע ארכיטעקטור.

#Microsoft Phi-4 5 (4:44:54)

א באווייז אז בעסערע דאטא שלאגט מער פאראמעטערס: א מאדעל פון 14 ביליאן (אויפן ווידעא געזאגט 16) וואס קאנקורירט מיט 70 ביליאן אויף רעזאנירונג. די אלטע חכמה איז געווען: מער פאראמעטערס, מער דאטא, בעסערער מאדעל. Microsoft האט געפרעגט: וואס אויב מען טרייניירט אויף בעסערע דאטא אנשטאט מער דאטא? זיי האבן געניירירט סינטעטישע ביישפילן פון הויכער קוואליטעט, שטארק אויסגעפילטערט, און באקומען א מאדעל וואס טוט זאכן וואס 70B מאדעלן ראנגלען זיך מיט. נישט קיין כישוף, נאר א באווייז אז קוואליטעט איז וויכטיגער ווי קוואנטיטעט. די Phi-3.5 MoE ווערסיע גייט נאך ווייטער: 42 ביליאן פאראמעטערס, אבער נאר 6.6 ביליאן אקטיוו אויף איינמאל.

#Falcon 3 6, און אנהייב פון #Command R 7 (4:45:55)

Falcon 3 איז געבויט צו לויפן אויף געווענליכע לעפטאפס, למשל א MacBook Air . אנטוויקלט דורך Technology Innovation Institute (TII) אין אבו דאבי, פריי פאר פארשונג און קאמערציעלע באנוץ. דער קורס מיינט אז עס פארדינט מער אויפמערקזאמקייט. די Mamba ווערסיע באנוצט state space models (SSM) אנשטאט טראנספארמערס: זיי באארבעטן לאנגע סעקווענצן מער עפעקטיוו, אבער זענען שוואך ביי קורצע פראגעס. עס ווערט געלויבט אז TII טענהט נאר 4 שפראכן אנשטאט צו מאכן זיך אז עס שטיצט 50 שפראכן אין שלעכטער קוואליטעט; ערליכע באגרעניצונגען העלפן פראגראמירער מער ווי פאלשע פעאיגקייטן.

דער נעקסטער מאדעל איז Command R פון Cohere, א "מייסטער" פאר RAG און באנוץ פון געצייג אויף פירמע-שטאפל, מיט א הויפט-מאדעל פון 104 ביליאן פאראמעטערס.

אפן-סאורס מאדעלן, געצייג, קאסטן און ריזיקעס (4:46:57)

Command, StarCoder, Qwen3 און Yi (4:46:57)

דער קורס פירט ווייטער מיט דער רשימה פון די בעסטע אפן-סאורס מאדעלן. אויסגערעכנט ווערט, אז איינער פון די פריערדיגע מאדעלן איז שטארק אין קאמפליצירטע לאגישע טראכטן, בעסער ווי רוב אנדערע.

Command איז דער ביישטייער פון דער פירמע Cohere צו דער וועלט פון "אפענע וואג" (open weights) מאדעלן. עס איז געבויט ספעציעל פאר גרויסע פירמעס, פאר .(Retrieval Augmented Generation) RAG RAG מיינט אז מען פארבינדט דעם מאדעל מיט די דאקומענטן פון דער פירמע, און ער ענטפערט פראגעס לויט זיי. ס'רוב מאדעלן זאגן אז זיי קענען דאס, אבער Command טוט עס באמת. ער באהאנדלט גרינג און רואיג API-פעלערן ביים באניצן געצייג (tools). ער שרייבט ערליך אין זיין דאקומענטאציע וואס זענען זיינע לימיטן ביי פארשידענע שפראכן, און טוט נישט פארשטעלן אז אלע שפראכן ארבעטן גלייך גוט. דער ליצענץ באגרעניצט פרייע קאמערציעלע באנוץ, אבער פאר פארשונג און אויספרובירן איז עס אינגאנצן אפן.

StarCoder איז דער "פראגראמירער'ס פראגראמירער" מאדעל. עס איז געבויט געווארן דורך BigCode, פון פראגראמירער פאר פראגראמירער. דער טרייניירונג-פראצעס איז עפנטלעך דאקומענטירט, און די דאטא-סעטס זענען אפן אויסגערעכנט. ער קען איבער 600 פראגראמירונג-שפראכן, אריינגערעכנט ,Haskell COBOL און Fortran פאר אלטע סיסטעמען. StarCoder ווערט באטראכט ווי דער "גאלדענער סטאנדארט" פאר אפן-סאורס מאדעלן וואס זענען פאקוסירט אויף קאוד. די BigCode גרופע האט אלעס דאקומענטירט: פון

וואו די טריינירונג-דאטא שטאמט, די עטישע באטראכטונגען, און פון וואס דער דאטא-סעט באשטייט. גארנישט איז באהאלטן.

א באזונדערע אייגנשאפט איז "fill-in-the-middle" (אויספילן אין מיטן). ס'רוב מאדעלן קענען נאר וועיטער שרייבן קאוד פון סוף. StarCoder קען אבער עראטן וואס דארף קומען אין מיטן פון קאוד וואס איז שוין דא. אזוי ארבעט באמת קאוד-אויספילונג אין דער פראקטיק. 600 שפראכן קלינגט ווי צופיל, ביז א גרויסער קליענט ברענגט א 30 יאר אלטע COBOL קאוד-באזע וואס דארף מען מאדערניזירן.

Qwen3 פון Alibaba ווערט באשריבן ווי די גלאבאלע "קראפט-סטאנציע" מיט א גלענצנדיקן "טראכטן-מאדוס". די גרייסן גייען פון בערך 0.6 ביליאן ביז 235 ביליאן פאראמעטערס, סיי "dense " און סיי MoE (Mixture of Experts) ארכיטעקטור. ער איז טריינירט געווארן אויף 36 טריליאן טאָקענס, צוויי מאל אזויפיל ווי די פריערדיגע ווערסיע. די אייגנשאפט וואס טוישט באמת ווי פראגראמירער טראכטן וועגן AI סיסטעמען איז דער "היברידער טראכטן-מאדוס": איין מאדעל קען טוישן מאדוס ווען מען וויל. ביי א שווערע פראבלעם צינדט מען אן "chain of thought": דער מאדעל ארבעט דורך די פראבלעם שריט נאך שריט איידער ער ענטפערט. ביי א פשוטע פראגע לעשט מען עס אויס, און מען באקומט גלייך א תשובה, אן פארלוירענע רעכן-קראפט. DevOps מאנשאפטן ניצן דאס צו שיקן יעדע פארלאנג לויט ווי שווער עס איז, און אזוי שניידן זיי שטארק אראפ די קאסטן פון inference. דער Apache 2.0 ליצענץ גילט פאר אלע גרייסן, און מען דארף מער נישט טשעקן וועלכע וועריסיע פאדערט וועלכן ליצענץ. דאס ווערט אנגערופן "א פרישער אטעם".

דער לעצטער מאדעל איז Yi, געבויט פון 01.AI ספעציעל פארן כינעזישן מארק. ער איז א צוויי-שפראכיקער ספעציאליסט, ענגליש און כינעזיש, און ביידע שפראכן באקומען די זעלבע אויפמערקזאמקייט ביים טריינירן. Yi איז דער מאדעל פאר יעדן וואס בויט ערנסט פאר כינעזיש-רעדנדיקע באניצער. רוב מאדעלן וואס זאגן אז זיי קענען פיל שפראכן רעדן ענגליש, און אלע אנדערע שפראכן ווערן צוגעלייגט ווי א נאכגעדאנק, אזוי אז די קוואליטעט פאלט שטארק אראפ נאך ענגליש. Yi איז טריינירט געווארן אויף ביידע מיט די זעלבע זארגפאלטיקייט, און מען שפירט דעם חילוק. דער context window פון 200,000 טאָקענס איז ברייט. די לעקציע פון Yi 1.5 איז אינטערעסאנט: מען האט נישט געטוישט די ארכיטעקטור, נאר מען האט געגעבן דעם באזע-מאדעל נאך 500 ביליאן טאָקענס פון קוואליטעט דאטא. דער מוסר השכל: בעסערע דאטא איז וויכטיגער ווי ארכיטעקטור-קונצן.

דער אמת וועגן בענטשמארקס: פארוואס טאבעלעס קענען אפנארן (4:50:02)

נאך די צען מאדעלן ברענגט דער קורס עפעס וואס איז גלייך וויכטיג: ווי אזוי צו טראכטן מיט א קריטישן קאפ וועגן בענטשמארק-צאלן און די ראנגליסטעס (leaderboards) וואס מען זעט אומעטום. Hugging Face'ס Open LLM Leaderboard לאזט גלייכע טעסטן לויפן אויף אלע מאדעלן. MMLU (Massive Multitask Language Understanding) איז איינער פון די מערסט ציטירטע טעסטן. זיי זענען נוצלעך צו פארגלייכן גלייכס מיט גלייכס.

אבער די פראבלעם איז, אז זיי מעסטן קינסטלעכע אויפגאבן וואס פאסן נישט שטענדיק צו אמת'ע ארבעט. א מאדעל וואס האט די העכסטע צאל אין MMLU קען נאך אלץ אויסטראכטן נעמען פון קונים אין אייער אמת'ן chatbot. די ראנגליסטע נעמט אן איינגאבעס פון יעדן איינעם, און דערפאר קען מען מאדעלן "אויסשארפן" ספעציעל צו געווינען אויפן בענטשמארק, אנשטאט זיי זאלן זיך פירן נוצלעך אין דער אמת'ער וועלט. דאס

האט מען שוין געזען ביי בענטשמארקס פאר בילדער-דערקענונג: הויכע צאלן, אבער דורכפאלן אין דער אמת'ער וועלט.

די פראקטישע עצה פונעם קורס:

- ניצט ראנגליסטעס ווי א אנהייב-פונקט, נישט ווי א לעצטע תשובה.
- דערנאך טעסט מיט אייער אייגענער דאטא: לאזט דעם מאדעל לויפן אויף טיפישע פראגעס פון אייערע אמת'ע באנוצן.
- מעסט די) latencyווי לאנג עס נעמט צו ענטפערן) אונטער דער לאסט וואס איר ערווארט.
- ציילט וויפיל אויסגעטראכטע ענטפערס (hallucinations) עס זענען אויף יעדע טויזנט ענטפערס אין אייער געביט.

קינסטלעכע בענטשמארקס וועלן אייך נישט זאגן צי א מאדעל ארבעט פאר אייער ספעציפישן פאל. דער בעסטער מאדעל איז דער וואס ארבעט בעסטנס פאר דעם וואס איר ווילט טון, נישט דער מיט דער העכסטער צאל אין דער טאבעלע.

די וויכטיגסטע געצייג צו לויפן און באהאנדלען מאדעלן (4:51:34)

די מאדעלן זענען נאר האלב פון דער מעשה. מען דארף געצייג זיי צו לויפן, צופאסן און ארויסשטעלן אין באנוץ. די אפן-סאורס וועלט האט געבויט א שטארקע געצייג-קאסטן אויף יעדן שטאפל.

לויפן מאדעלן אויפן אייגענעם קאמפיוטער:

- Ollama: מיט דריי באפעלן איז מען פארטיג. דאס איז דער גרינגסטער וועג צו לויפן אפן-סאורס LLMs.
- Hugging Face: דער "GitHub פון AI", וואו עס געפינען זיך איבער 325,000 מאדעלן. דאס איז דער גרעסטער שפייכלער פון אפן-סאורס מאדעלן, און יעדער מאדעל וואס איז דערמאנט געווארן איז דארט צו געפינען.
- LM Studio, GPT4All און Jan: אפליקאציעס מיט א גראפישן אויבערפלאך (GUI) פאר מענטשן וואס זענען נישט טעכניש. LM Studio איז בכלל דער בעסטער: א מאדעל-בלעטערער, א שמועס-אויבערפלאך, א לאקאלער סערווער און פרטים'דיקע קאנטראלן. GPT4All איז די פשוטסטע ברירה: אראפלאדן, אינסטאלירן, שמועסן, און עס ארבעט אינגאנצן אן אינטערנעט. Jan איז פאקוסירט אויף פריוואטקייט: אלע דאטא בלייבט לאקאל, גוט פאר סענסיטיווע באנוצן.

געצייג פאר) fine-tuningנאך-טריינירן א מאדעל):

- Unsloth: מען קען נאך-טריינירן מאדעלן פון 70 ביליאן פאראמעטערס אויף איין געווענליכן GPU פאר קונים. עס קען נאך-טריינירן Llama, Mistral, Qwen און DeepSeek אויף GPUs וואס ס'רוב מענטשן פארמאגן שוין.
- LLaMA-Factory: מען קען נאך-טריינירן יעדן גרויסן מאדעל דורך א וועב-אויבערפלאך, אן קיין קאוד.

בויען און ארויסשטעלן:

- LangChain און LlamaIndex: פארבינדן אייערע LLMs מיט אייער דאטא און בויען אמת'ע אפליקאציעס. מען קען פארבינדן יעדן LLM מיט PDFs, דאטא-באזעס, וועבזייטלעך און APIs, און בויען .RAG LangChain איז א ראם פאר אלגעמיינע אפליקאציעס, וואו מען קען שאפן אגענטן, קייטן (chains), זכרון

און באנוץ פון געצייג. LlamaIndex איז ספעציאליזירט אויף אריינברענגען און ארויסזוכן דאטא, אויסגעשארפט פאר RAG-ליניעס. ביידע ארבעטן מיט Llama מאדעלן און מיט Hugging Face מאדעלן, און זיי זענען דער סטאנדארט געצייג-קאסטן פאר LLM-אפליקאציעס אין פראדוקציע.

• n8n: א וויזועלער בויער פון ארבעט-גענג. מען קען שלעפן און אראפלייגן AI אגענטן אן קאוד, בויען ארבעט-גענג פון כמה שריט וויזועל, און פארבינדן LLMs מיט יעדן API אדער דאטא-באזע.

• vLLM און SGLang: געצייג פאר שנעלע באדינונג אין פראדוקציע. vLLM שטעלט אפן-סאורס מאדעלן צו ווי א שנעלער API, וואס קען דירעקט פארבייטן דעם OpenAI API פארמאט.

• IBM watsonx : אויפזיכט און קאנטראל פאר גרויסע פירמעס צוואמען מיט אפן-סאורס מאדעלן. עס האלט Llama 4 און IBM'ס אייגענע Granite מאדעלן, מיט אויפזיכט אויפן שטאפל פון גרויסע פירמעס.

וועלכע הארדווער דארף מען? (4:54:10)

דער קורס ענטפערט די פראגע וואס ווערט מערסטנס געפרעגט: וויפיל זכרון (RAM/VRAM) דארף מען?

• 4GB: לויפט קליינע מאדעלן שלעכט. נישט רעקאמענדירט פאר עפעס פראקטיש.

• 8GB: באהאנדלט מאדעלן פון 3 ביז 7 ביליאן פאראמעטערס גאנץ גוט, ווי Mistral 7B, Llama 3 8B, Gemma 4B.

• 16GB: עפנט רוב ברירות און לויפט רוב פאפולערע מאדעלן באקוועם.

• 32GB און מער: לאזט לויפן גרויסע מאדעלן אן וויטאגדיקע פארשלעפונגען.

• 80GB VRAM און מער (A100, H100): לויפט 70 ביליאן מאדעלן אינגאנצן, אדער מאדעלן פון 235 ביליאן און מער מיט quantization.

צוויי וויכטיגע ערקלערונגען: דער זכרון אויפן GPU (VRAM) איז וויכטיגער ווי דער CPU-זכרון פאר די שנעלקייט פון inference. און quantization מיינט אראפצונעמען די געניקייט פונעם מאדעל, פון 32 ביט אויף 4 אדער 8 ביט. דאס שניידט בערך אראפ האלב פונעם זכרון וואס מען דארף, מיט ווייניג פארלוסט אין קוואליטעט.

וויפיל קאסט עס צו לויפן א LLM? (4:55:11)

מען זאגט אז אפן-סאורס איז אומזיסט, אבער דאס איז נאר האלב אמת. די פרייזן:

• לאקאל, אויף אייער אייגענער הארדווער: אומזיסט, אויב די הארדווער איז שוין גוט גענוג. מען דארף אבער צולייגן די עלעקטרע-קאסטן ביי שטענדיגער באנוץ.

• א Cloud VPS נאר מיט CPU, אן GPU: 20 ביז 50 דאלער א חודש. גוט פאר קליינע מאדעלן, מיט פאמעלעכערן inference.

• Cloud GPU סערווערס: בערך 50 ביז 200 דאלער און מער א חודש, לויט דעם GPU-שטאפל און ווי פיל מען ניצט.

• באדינטע פלאטפארמעס ווי watsonx און AWS Bedrock: ענליך צו OpenAI'ס פרייזן, אבער מיט אייגנטום אויפן מאדעל און מיט מעגלעכקייטן פאר הלכות פון געזעצן (compliance).

• DeepSeek API: בערך 0.28 דאלער פאר א מיליאן אריינגאבע-טאָקענס, און 0.42 דאלער פאר א מיליאן ארויסגאבע-טאָקענס. דאס איז א קליינער חלק פון GPT'ס פרייז.

די באהאלטענע קאסטן וואס מען פארגעסט: עלעקטרע ביי לאקאלן לויפן, די צייט פון אינפראסטרוקטור-אינזשענירן ביי זיך-אליין האלטן, און רעזערוו-סיסטעמען וועלן די הויפט-סיסטעם פאלט אוועק. פעסטע סערווער-קאסטן זענען בעסער ווי טוישנדיגע API-קאסטן וועלן אייער באנוץ איז פאראויסצוזען און וואקסט. די חשבון קערט זיך געוועלדיגט ארום ביי בערך 50,000 ביז 100,000 API-רופן א חודש, לויט דער גרייס פון די פראגעס. פאר אויספרובירן און ביי ווייניג באנוץ זענען די באצאלטע APIs אפט ביליגער, וועלן מען רעכנט אריין אייער צייט. אבער אין פראדוקציע מיט א גרויסע צאל פראגעס געווינט זיך-אליין-געהאלטענער אפן-סאורס אויף קאסטן. די עצה: רעכנט אויס די צאלן פאר אייער באנוץ איידער איר באשליסט.

די ריזיקעס (4:56:14)

דער קורס באטאנט אז אמת'ע אחריות מיינט אויך צוגעבן די שלעכטע זייטן:

- אויסגעטראכטע ענטפערס (hallucination): יעדער LLM, סיי אפן-סאורס סיי באצאלט, קען זאגן מיט פולן בטחון זאכן וואס זענען אינגאנצן פאלש. ער קען ציטירן א ארטיקל וואס עקזיסטירט נישט, געבן פאלשע מעדיצינישע עצות מיט פולער זיכערקייט, אדער אויסטראכטן א פריערדיגן געריכט-פסק. ביי יעדער וויכטיגער זאך מוז א מענטש איבערקוקן די ענטפערס. דאס איז נישט קיין ברירה.
- פארטייאישקייט (bias): מאדעלן טריינירט אויף אינטערנעט-דאטא ירשענען די פארטייאישקייט פונעם אינטערנעט. אויב די טריינירונג-דאטא שטעלט פאר צופיל געוויסע שטאנדפונקטן, גרופעס אדער שפראכן, שפיגלט דער מאדעל דאס אפ. דאס איז א ערנסטערע זארג ביי געוויסע אפן-סאורס מאדעלן, וואו מען קלייבט נישט אויס די דאטא אזוי געוי וי ביי לאבארטאריעס מיט גרויסע מיטלען. מען דארף קענען די טריינירונג-דאטא פון אייער מאדעל איידער מען ניצט אים פאר באשלוסן וואס רירן אן מענטשנס לעבן.
- הסכמה און אויפזיכט אויף דאטא: איז די טריינירונג-דאטא געזאמלט געווארן מיט די מענטשנס הסכמה? פאסט דער מאדעל צו די געזעצן פון דאטא-שוץ אין אייער געגנט? IBM'ס Granite מאדעלן פארעפנטלעכן פונקטלעך וועלכע דאטא איז געוצט געווארן. אזא אפנקייט איז ווערטפול ביי א compliance-אונטערזוכונג, און ס'רוב גיבן דאס נישט. מען דארף וויסן וואס מען שטעלט ארויס, בפרט אין רעגולירטע אינדוסטריעס.
- זיכערהייט: אפן-סאורס האט אייגנארטיגע שוואכקייטן. ווייל די וואגן (weights) פונעם מאדעל זענען עפנטלעך, קענען אטאקירער פרובירן דעם מאדעל אויף שוואכקייטן אן קיין באגרעניצונג. prompt injection, דאטא-פארגיפטונג (data poisoning), און model inversion אטאקעס (ארויסציען טריינירונג-דאטא פונעם מאדעל) ווערן אלע גרינגער ווען די וואגן זענען אפן. מען איז אליין פראנטווארטליך פאר דער פארטייידיגונג: קאנטראלירן אריינגאבעס, באגרעניצן ווי אפט מען מעג רופן, און נאכקוקן אויף אומגעוועלכע אקטיוויטעט. קיינס פון די זאכן פארמיידט א פעסט-באשלאסענעם אטאקירער, אבער אלע מאכן אטאקעס שווערער.
- טעכנישע קאמפליצירטקייט: מען זאל זי נישט אונטערשאצן. אייגענע מאדעלן לויפן פאדערט אינפראסטרוקטור, טעכנישע מומחיות און שטענדיגע אויפהאלטונג. מאדעלן דארפן דערהיינטיגונגען, מען מוז אנלייגן זיכערהייט-פלאסטערס, און מען מוז אויפשטעלן נאכקוקן אויף דער לייסטונג. עס איז נישט אזוי פשוט ווי א API-שליסל. פאר נישט-טעכנישע ארגאניזאציעס עקזיסטירן פלאטפארמעס ווי watsonx.ai פונקט צוליב דעם. ווער עס האלט אליין דעם מאדעל, זאל אריינרעכענען אין בודזשעט אינזשעניר-צייט, נישט נאר הארדווער.

צום שלוס פונעם טייל ווערט אונטערגעשטראכן: AI ווערט מער נישט קאנטראלירט נאר דורך גרויסע פירמעס. מיט אפן-סאורס LLMs קענען סטודענטן, פראגראמירער, נייע פירמעס און געשעפטן לויפן AI אויף זייערע אייגענע סיסטעמען, באשיצן זייער דאטא, אראפשניידן קאסטן, און בויען אמת'ע AI פראדוקטן מיט מער קאנטראל.

דער זכרון פון א LLM און דער context window (4:58:45)

א נייער טייל הייבט זיך אן מיט א משל: צוויי מענטשן שמועסן וועגן א טעמע. ווי דער שטייגער פון מענטשליכן מוח, ווערט די טעמע פארגעסן נאך א קורצע צייט, און מען קען זיך נישט דערמאנען סיידן מען ווערט דערמאנט. דאס זעלבע קען מען ערווארטן ביים רעדן מיט א LLM. ער האט אויך א זכרון: וועט מען שמועסט מיט אים, ווערט דער שמועס געהאלטן אין דעם אקטיוון זכרון, אויך גערופן "working memory". כל זמן דער שמועס בלייבט אין דעם פארהאנענעם זכרון-פלאץ, וועט מען נישט טרעפן אויסגעטראכטע ענטפערס, און דער LLM וועט נישט פארגעסן קיין פרט. וועט מען גייט אריבער די גרענעץ, קען מען ערווארטן דאס זעלבע ווי ביים מענטש. מען קען עס אויך אזוי באטראכטן: ווי לאנג קען א LLM פירן דעם שמועס איידער ער פארלירט דעם זכרון, הייבט אן אויסטראכטן זאכן, אדער ווערט "באָרד" און פירט נאר קליינע שמועסן.

ערשט נעמט מען א קורצן שמועס וואס בלייבט אין די גרענעצן פונעם זכרון. א קעסטל ווערט געוויזן וואס שטעלט פאר דעם context window, און דער גאנצער שמועס פאסט אריין. דאס הייסט, וועט דער LLM האט געשאפן א תשובה, איז אלעס געווען אין זיין זכרון. מען שטעלט א פראגע, באקומט א תשובה, פרעגט א צווייטע פראגע וואס איז פארבונדן מיט דער ערשטער, און באקומט די לעצטע תשובה. אלעס גוט.

דערנאך נעמט מען א לענגערן שמועס: מען שיקט א פראמפט, באקומט א תשובה, און מען גייט אהין און צוריק נאך עטליכע מאל. ביי א געוויסן פונקט ווערט דער שמועס לענגער ווי דער context window פונעם מאדעל. דאס הייסט, די פראגעס און תשובות פונעם אנהייב פונעם שמועס זענען נישט שוין נישט פארהאן פארן מאדעל. ער האט קיין זכרון פון זיי וועט ער שאפט נייע תשובות. דער LLM קען פרובירן זיך צו באגיין אן זיי.

קאנטעקסט-פענצטער און די יסודות פון פראמפט-אינזשעניריע (5:01:20)

פארוואס עס פעלט קאנטעקסט, און וואס איז א טאָקען (5:01:20)

דער קורס ערקלערט אז וועט דער מאדעל ווייסט נישט פונקטליך וואס מען וויל, טרעט ארויף דער באגריף "האלוצינאציע". פונקט ווי א מענטש, דארף דער LLM אז מען זאל אים איבערלאזן אנווייזונגען. אן זיי מאכט ער עפעס ווי "געבילדעטע טרעפונגען", און דאס קען ברענגען ענטפערס וואס זענען אינגאנצן נישט שייך צום ענין. דערפאר איז וויכטיג צו פארשטיין ווי דער קאנטעקסט-פענצטער ארבעט, כדי ארויסצוקריגן דאס מערסטע פונעם מאדעל.

יעדער קאנטעקסט-פענצטער ווערט געמאסטן מיט א מאָס וואס הייסט tokens. מען קען עס באטראכטן ווי א איינהייט וואס מעסט די גרייס פונעם איצטיגן אדער "ארבעטס-זכרון". ביי מענטשן איז די קלענסטע איינהייט אינפארמאציע וואס מען ניצט צו פארשטעלן שפראך געווענליך א איינציגער צייכן: א אות, א וואָרט, א נומער, א פונקטואציע-צייכן א.א.וו. ביי א LLM אבער איז די קלענסטע איינהייט א טאָקען. א טאָקען קען זיין א איינציגער צייכן, א טייל פון א וואָרט, א גאנץ וואָרט, אדער אפילו א קורצע פראזע פון עטליכע ווערטער.

אין דער לעקציע ווערט געוויזן א ביישפיל: דער זאץ "hi welcome to Simplilearn" צעטיילט אין קעסטלעך. דאס ערשטע קעסטל, "hi", איז א טאָקען. דאס צווייטע, "welcome", איז אויך א טאָקען, און אזוי ווייטער.

דאס געצייג וואס פארוואנדלט שפראך אין טאָקענס הייסט tokenizer. פארשידענע tokenizers קענען צעטיילן דעם זעלבן טעקסט אויף פארשידענע אופנים. אבער אין אלגעמיין שטעלט א געווענליך ענגליש ווארט פאר בערך 1.5 טאָקענס. למשל, 1,000 ווערטער קענען ארויסגעבן בערך 1,500 טאָקענס. יעדער LLM האט זיין אייגענעם שטייגער ווי צו צעטיילן די אריינגאבן און ארויסגאבן פונעם באניצער, און זיי האבן אויך פארשידענע גרייסן קאנטעקסט-פענצטער.

ווי אזוי א LLM באשטימט די לענג פונעם קאנטעקסט (5:03:27)

נאכדעם וואס מען פארשטייט וואס טאָקענס זענען, ערקלערט דער קורס ווי אזוי א LLM פראצעסירט זיי אינערהאלב דעם קאנטעקסט-פענצטער. Transformer-מאדעלן ניצן א מעכאניזם וואס הייסט self-attention. דער מעכאניזם רעכנט אויס די באציאונגען און אפהענגיקייטן צווישן פארשידענע טיילן פון דער אריינגאב, למשל צווישן ווערטער אינעם אנהייב און אינעם סוף פון א פאראגראף. Self-attention רעכנט אויס וועקטארן פון "וואגן" (weights). יעדע וואג וועזט ווי שייך א געוויסער טאָקען איז צו אלע אנדערע טאָקענס אויף די וועיטערדיגע פלעצער. די גרייס פונעם קאנטעקסט-פענצטער באשטימט די מאקסימאלע צאל טאָקענס וואס דער מאדעל קען "באטראכטן" מיט אמאל.

די גרייסן פון קאנטעקסט-פענצטער זענען שנעל געוואקסן:

- די ערשטע LLMs האבן געהאט בערך 2,000 טאָקענס.
- Gemini Flash ניצט היינט בערך 1 מיליאן טאָקענס.
- עטליכע מאדעלן קענען האנדלען מיט נאך מער. למשל, OpenAI איז ארויסגעקומען מיט GPT-4.1, וואס גלייכט זיך צום שיעור פון Gemini, דאס הייסט איבער 1 מיליאן.

אזויפיל טאָקענס וואלטן דערלויבט צו רעדן מיטן LLM כמעט אן א סוף, 24 שעה אין איינעם. פארוואס דארף מען אבער אזויפיל טאָקענס? דער קורס זאגט אז דא איז דא א געוויסער "גרויער שטח" אין די טעכנישע פרטים. כדי דאס צו פארשטיין, מוז מען זען ווי אזוי א LLM ניצט זיין קאנטעקסט-פענצטער.

וואס פילט אן דעם קאנטעקסט-פענצטער (5:04:59)

עס ווערט ערקלערט אז דער קאנטעקסט-פענצטער ווערט נישט אנגעפילט נאר מיט די אריינגאבן פונעם באניצער און די ענטפערס פונעם מאדעל. אין אים געפינען זיך אויך:

- א system prompt, אפט באהאלטן פונעם באניצער, וואס פירט אן דאס פירונג פונעם מאדעל.
- בילדער וואס באניצער לאדן ארויף. דער קורס דערמאנט דא דעם "Ghibli"-שטיל טרענד.
- פאטאגראפיעס, ווידעאס, PDF-טעקעס, טעקסט-דאקומענטן און אפילו מקור-קאוד. דער מאדעל באציט זיך צו דעם אלעם ביים ענטפערן.
- דרויסנדיגע דאטא, וואס דער LLM שטיצט זיך אויף איר פאר די רעזולטאטן. דאס ווערט געוועט פאר retrieval augmented generation, באקאנט ווי RAG. דאס אלעס פארנעמט אויך פלאץ אינעם פענצטער.

דערפאר, מיט בלויז עטליכע לאנגע דאקומענטן אדער שטיקלעך קאוד, קען דער קאנטעקסט-פענצטער זיך גאר שנעל אנפילן. עטליכע באניצער האבן אפשר שוין געזען א מעלדונג פון ChatGPT וואס הערט זיך ווי "געווג פאר היינט, קומט צוריק מארגן פאר די איבעריגע ענטפערס".

די שווערִיגקייטן פון לאנגע קאנטעקסט-פענצטער (5:06:03)

מען קען מיינען אז וואס גרעסער דער קאנטעקסט-פענצטער, אלץ בעסער. דער קורס באמערקט אבער אז מיט גרויס כח קומט גרויס אחריות, און די LLMs שטויסן זיך אן אין שווערִיגקייטן:

- רעכן-קאסטן: דאס איז די ערשטע און קלארסטע שווערִיגקייט. די רעכן-פאדערונגען פונעם self-attention מעכאניזם וואקסן קוואדראטיש מיט דער לענג פון דער אריינגאב. פשוט געזעדט: וועז די צאל טאָקענס טאפלט זיך, וועזט די נויטיגע רעכן-קראפט פיר מאל אזוי גרויס. די סיבה איז אז דער מאדעל באטראכט די באציאונגען צווישן יעדן נייעם טאָקען און יעדן פריערדיגן טאָקען אין דער סעקווענץ.
- לייסטונג: לאנגע פענצטער ברענגען אויך פראבלעמען מיט דער לייסטונג. ווי מענטשן, ווערן LLMs איבערגעפילט פון צופיל פרטים און קענען אנהייבן נעמען "קורצע וועגן" אינעם טראכטן. א שטודיע פון 2023 האט געפונען אז מאדעלן ארבעטן דאס בעסטע וועז די וויכטיגע אינפארמאציע שטייט אינעם אנהייב אדער אינעם סוף פון דער אריינגאב. די לייסטונג פאלט אראפ וועז דער מאדעל דארף פראצעסירן שליסל-אינפארמאציע וואס ליגט אין מיטן פון לאנגע סעקווענצן.
- זיכערהייט: דאס איז די לעצטע און וויכטיגע שווערִיגקייט. א לענגערער קאנטעקסט-פענצטער שאפט א ברייטערע "אטאקע-פלאך" פאר שעדליכע פראמפטס. בייווילִיגער אינהאלט קען זיין באהאלטן טיף אינעווייניג אינעם טעקסט.

איבערבליק פונעם טייל וועגן פראמפט-אינזשעניריע (5:07:37)

דאָ הייבט זיך אן א נייער טייל פונעם קורס, וועגן פראמפט-אינזשעניריע. מען לערנט וואס איז איר באדייט, און ווי אזוי סטרוקטורירטע פראמפטס פארבעסערן די ארבעט מיט AI. דער פלאן פונעם טייל:

- די עיקר-טיילן פון א גוטן פראמפט: קלארקייט, קאנטעקסט, ספעציפישקייט, באגרענעצונגען און ביישפילן. זיי זארגן פאר פונקטליכע און שייכותדיגע רעזולטאטן.
- אמת'ע אנווענדונגען פון פראמפט-מוסטערן: ווי פראמפט-אינזשעניריע פארבעסערט שאפן אינהאלט, הילף ביים קאוד, קונה-שטיצע און דאטא-אנאליז.
- פארשידענע פראמפט-טעכניקן: אנווייזונג-באזירטע, ביישפיל-געטריבענע, "אויספילן די ליידיגע ערטער", און שריט-ביי-שריט פראמפטס.
- געוועגליכע טעותים: אומקלארע אנווייזונגען, איבעריגע פרטים און שלעכטע פארמאטירונג, צווזאמען מיט פראקטישע איבונגען.
- פארגעשריטענע טעכניקן: zero-shot, few-shot און chain-of-thought פראמפטינג, וויכטיג פאר קאמפליצירטע פראבלעמען און פאר מער פינקטלעכקייט.

דער ציל איז אז מען זאל קענען שרייבן שטארקע, סטרוקטורירטע פראמפטס וואס ברענגען ארויס דאס מערסטע פון AI אין פארשידענע געביטן.

וואס איז פראמפט-אינזשעניריע (5:09:08)

דער קורס ערקלערט אז פראמפט-אינזשעניריע איז א וויכטיגע פעאיגקייט אין דער וועלט פון AI, בפרט ביי ארבעט מיט גרויסע שפראך-מאדעלן ווי ChatGPT. זי באשטייט פון אויסארבעטן גוט-סטרוקטורירטע פראמפטס, כדי אנצופירן די AI-ענטפערס אויף א ווירקזאמען אופן, צי פאר שאפן אינהאלט, אויטאמאציע

אדער לייזן פראבלעמען. א גוט-אויסגעארבעטער פראמפט פארזיכערט א פונקטליכן און שייכותדיגן רעזולטאט.

עס גייט נישט נאר אין שרייבן באפעלן, נאר אין קלאר מיטטיילן די כוונה צו דער AI. דער אופן ווי א פראמפט איז פארמולירט קען שטארק משפיע זיין אויפן ענטפער: וואס פונקטליכער און מער סטרוקטורירט, אלץ בעסער דער רעזולטאט. דערפאר דארפן געשעפטן, פראגראמירער און אינהאלט-שאפער אינוועסטירן צייט אין פארבעסערן זייער שרייבן פון פראמפטס.

די דעפיניציע: פראמפט-אינזשעניריע איז דער פראצעס פון פלאנירן און פארפיינערן פראמפטס, כדי צו באקומען די געווינטשענע ענטפערס פון AI-מאדעלן. ווי גוט דער AI-אינהאלט קומט ארויס, העננט שטארק אפ ווי גוט דער פראמפט איז סטרוקטורירט. וועז מען פארשטייט ווי שפראך-מאדעלן טייטשן די אריינגאבן, קען מען פארגרעסערן די ווירקזאמקייט פון AI.

א ביישפיל פון א AI-צ'עטבאט פאר קונה-שטיצע:

- א שלעכט-סטרוקטורירטער פראמפט: "Help the customer with their issue". דער ענטפער: "זיכער, ביטע גיט די פרטים פון דעם קונה'ס פראבלעם: מיט וועלכן פראדוקט אדער סערוויס, אנדערע מעלדונגען, אדער א שייכותדיגע באשטעלונג- אדער קאנטע-אינפארמאציע".
- א גוט-אויסגעארבעטער פראמפט: "Analyze the customer issue based on previous conversation history and suggest a step-by-step troubleshooting process before escalating to a human agent". דא אנאליזירט דער מאדעל דעם פראבלעם לויטן קאנטעקסט און די פרטים, און גיט א שריט-ביי-שריט פראצעס צו פארריכטן, ווי מען זעט אויפן בילד.

איצט האט די AI קלארע אנווייזונגען, און די שאנסן פאר א שייכותדיגן און פראקטישן ענטפער זענען גרעסער.

ווי איז ChatGPT (5:11:13)

ChatGPT איז א פארגעשריטענער AI שפראך-מאדעל, אנטוויקלט פון OpenAI. ער ניצט natural language processing (NLP) צו שאפן טעקסט-ענטפערס ענליך צו מענטשן. ער איז געבויט אויף דער GPT (generative pre-trained transformer) ארכיטעקטור, וואס דערמעגליכט אים צו פארשטיין, טייטשן און שאפן טעקסט לויט דער אריינגאב פונעם באניצער. אויפן בילד זעט מען א ביישפיל: מען פרעגט "what is machine learning in one line", און מען באקומט א ענטפער.

ChatGPT קען העלפן ביי פארשידענע ארבעטן: ענטפערן פראגעס, שרייבן אינהאלט, צוואמפאסן טעקסט, שאפן קאוד, און אפילו פירן אן אינטעראקטיוון שמועס. ער איז געבויט אויף דיפ-לערנען. ער זאגט פאראויס און שאפט טעקסט דורך אנאליזירן ריזיגע מאסן דאטא, און דערפאר איז ער זייער ווירקזאם פאר אויטאמאציע, "ברייסטאָרמינג" און לייזן פראבלעמען. ער שטיצט פארשידענע אינדוסטריעס: חינוך, קונה-שטיצע, שאפן אינהאלט און פראגראמירונג. דער מאדעל פארבעסערט זיך כסדר דורך fine-tuning און באמערקונגען פון באניצער, כדי צו געבן מער שייכותדיגע און פונקטליכע ענטפערס, סיי פאר פריוואטע און סיי פאר פראפעסיאנעלע באנוץ. לויטן קורס איז ער א שטארקער AI-געהילף וואס פארגרעסערט פראדוקטיווקייט און מאכט גרינגער די קאמוניקאציע.

די עיקר-טיילן פון א ווירקזאמען פראמפט: קלארקייט (5:12:45)

פראמפט-אינזשעניריע איז די פראקטיק פון פארזיכטיג אויסארבעטן אריינגאבן, כדי אנצופירן א AI-מאדעל צו פונקטליכע, שייכותדיגע און באדייטפולע ענטפערס. א גוט-סטרוקטורירטער פראמפט מאכט די ענטפערס מער פונקטליך, קורץ, און צוגעפאסט צו די ערוואַרטונגען פונעם באניצער. די שליסל-עלעמענטן זענען: קלארקייט, קאנטעקסט, ספעציפישקייט, באגרענעצונגען, און פארשידענע ביישפילן. יעדער פון זיי שפילט א וויכטיגע ראלע אין פארמירן דעם ענטפער.

קלארקייט: א פראמפט דארף זיין דירעקט, גרינג צו פארשטיין און אן צוויי-דייטיגקייט. וועז עס פעלט קלארקייט, קען די AI אומריכטיג פארשטיין די בקשה און געבן אומקלארע אדער נישט-שייכותדיגע ענטפערס. למשל, "tell me about AI" זאגט נישט צי דער באניצער וויל א היסטארישן איבערבליק, אנווענדונגען אדער טעכנישע פרטים. די AI קען דעריבער געבן א לאנגן, אלגעמיינעם ענטפער וואס פאסט נישט צו דער כוונה.

אנשטאט דעם קען מען שרייבן: "Provide a brief history of artificial intelligence, including key milestones from the 1950s to the present". פארוואס איז דאס בעסער?

- עס באשטימט קלאר דעם ראם: די היסטאריע פון AI.
- עס גיט א צייט-ליניע: פון די 1950ער ביז היינט.
- די פראזע "key milestones" לאזט די AI זיך קאנצענטרירן אויף באדייטנדיגע אנטוויקלונגען, אן איבעריגע פרטים.

קאנטעקסט און ספעציפישקייט (5:15:22)

קאנטעקסט: מען דארף געבן שייכותדיגע הינטערגרונט-אינפארמאציע. אן דעם קען די AI געבן א ענטפער וואס איז אדער צוברייט אדער נישט שייך צו דער כוונה פון דער פראגע. למשל, "summarize the article" זאגט גארנישט וועגן דער טעמע פונעם ארטיקל, זיין לענג אדער די הויפט-פונקטן. דער צוואמפאס קען דעריבער נישט ארויסברענגען וואס דער באניצער זוכט.

אנשטאט דעם: "Summarize the article on climate change, highlighting the causes and effects in under 200 words". פארוואס איז דאס בעסער?

- עס באשטימט די טעמע: קלימאט-ענדערונג.
- עס ווייזט אן די AI זיך צו קאנצענטרירן אויף סיבות און ווירקונגען, נישט אויף אלגעמיינע אינפארמאציע.
- עס באגרענעצט די צאל ווערטער און פארזיכערט א קורצן צוזאמפאס.

וועז מען לייגט צו קאנטעקסט-פרטים, פאסט די AI צו דעם ענטפער פונקט צו די באדערפענישן, שייכותדיג און סטרוקטורירט.

ספעציפישקייט: מען דארף קלאר באשטימען דעם ראם. דאס פארשמאלט דעם ענטפער, כדי ער זאל זיין קורץ און שייכותדיג, און נישט צו ברייט אדער אוועק פון דער טעמע. א אומקלארער פראמפט קען ברענגען א צעשפרייטן ענטפער, וואס פאדערט נאכדעם נאך דערקלערונגען אדער פארבעסערונגען. למשל, "list some programming languages" גיט דער AI קיין ריכטונג וועלכע סארט פראגראמיר-שפראכן מען דארף. זי קען דעריבער אויסקלייבן צופעליגע שפראכן, טייל פון זיי נישט שייך צו וואס דער באניצער דארף.

פראמפט-עלעמענטן אין פראקטיק און אנוועגדונגען (5:17:10)

ספעציפישקייט און באגרענעצונגען – די טעאריע (5:17:10)

דער קורס ענדיגט די ערקלערונג פון ספעציפישקייט מיט א ביישפיל. א שוואכער פראמפט פרעגט סתם נאך פראגראמירונג-שפראכן. א בעסערער פראמפט פרעגט: "נענן פינף פראגראמירונג-שפראכן וואס מען ניצט געווענליך פאר וועב-אנטוויקלונג, צוואמען מיט זייערע הויפט-אנווענדונגען".

פארוואס איז דער פראמפט בעסער?

- ער באגרענעצט די צאל שפראכן צו פינף, און אזוי איז דער ענטפער פאקוסירט.
- ער שטעלט פעסט או דער געביט איז וועב-אנטוויקלונג, אזוי קומען נישט אריין שפראכן וואס האבן נישט קיין שייכות.
- ער פרעגט נאך די הויפט-אנווענדונג, אזוי איז דער ענטפער מער אינפארמאטיוו און פראקטיש.

או מען שטעלט קלאר פעסט וויפיל דער ענטפער זאל אנכאפן, פאלט אוועק דאס טראכטן פון AI אויף גראדעוואל, און מען באקומט פונקטליך וואס מען דארף.

דער פערטער עלעמענט איז באגרענעצונגען (constraints). זיי קאנטראלירן דעם רעזולטאט דורך גרעניצן. זיי מאכן זיכער או דער ענטפער פאסט אריין אין דער פארלאנגטער צאל ווערטער, אין דעם פארמאט אדער אין דעם טאן. אן באגרענעצונגען קען דער ענטפער ארויסקומען צו לאנג, צו קורץ, אדער נישט פאסיג צום געוואונטשענעם סטיל.

א ביישפיל: דער פראמפט "ערקלער די Renaissance" זאגט נישט ווי דעטאלירט אדער ווי סטרוקטורירט די ערקלערונג דארף זיין. AI קען ארויסגעבן א לאנגן דעטאלירטן עסיי, וואס דער באניצער האט אפשר בכלל נישט געוואלט. אנשטאט דעם שרייבט מען: "שרייב א סיכום פון 100 ווערטער איבער דער Renaissance-תקופה אין א פארמאלן טאן".

פארוואס איז דאס בעסער?

- עס באגרענעצט די צאל ווערטער צו 100, און אזוי איז דער ענטפער קורץ און קלאר.
- עס שטעלט פעסט א פארמאלן טאן, אזוי קומט ארויס אן אקאדעמישער סטיל.
- עס פירט אן AI צו געבן א ענטפער וואס פאסט צו דער ערווארטעטער סטרוקטור.

באגרענעצונגען זענען באזונדערס נוצליך ביי ביזנעס, פארשונג און שאפן אינהאלט, וואו פונקטליכער פארמאט און סטיל זענען וויכטיג.

דעמא: קלארקייט אין ChatGPT (5:18:41)

אין דער לעקציע ווערט געוויזן א דעמא אין דער Plus-ווערסיע פון ChatGPT. מען גייט דורך איינס נאך דאס אנדערע אלע עלעמענטן וואס זענען שוין דורכגערעדט געווארן: קלארקייט, קאנטעקסט, באגרענעצונגען, ספעציפישקייט, און ביישפילן.

דער קורס באמערקט אז ווען מען ניצט די זוך-פונקציע, זוכט ChatGPT אויפן וועב. אן דעם גיט עס א ענטפער פון דעם וואס עס איז טרייניירט געווארן.

ערשטנס קלארקייט. מען שרייבט "Tell me about AI". דער ענטפער ערקלערט אז AI באדייט נאכמאכן מענטשליכע אינטעליגענץ אין מאשינען, אזוי אז זיי קענען טון ארבעטן וואס פאדערן געוועלך א מענטשליכן שכל. עס קומען דערביי די סארטן AI, ווי AI ארבעט, געוועלכע אנווענדונגען, און די צוקונפט פון AI. דאס איז אלגעמיין און נישט פאקוסירט.

דערנאך שרייבט מען קלאר וואס מען ווילּ: "גיב די היסטאריע פון קינסטליכער אינטעליגענץ, אריינגערעכנט די הויפט-מיילשטיינער פון די 1950ער יארן ביז היינט". דא ווערט קלאר באשריבן וויפיל דער ענטפער זאל אנכאפן: די היסטאריע פון AI. עס ווערט אויך באשריבן די צייט, פון די 1950ער ביז היינט. דער אויסדרוק "הויפט-מיילשטיינער" העלפט AI זיך קאנצענטרירן אויף באדייטנדע אנטוויקלונגען אן איבערגע פרטים.

דער רעזולטאט גייט לויט די יארן:

- די 1950ער, די געבורט פון AI: אין 1950 שטעלט Alan Turing פאר דעם Turing Test אין זיין ארטיקל "Computing Machinery and Intelligence". די Dartmouth-קאנפערענץ, אנגעפירט פון John McCarthy, האט אפיציעל געמטבע'ט דעם טערמין "artificial intelligence".
- די 1960ער און 1970ער: וויטערדיגע אנטוויקלונגען.
- די 1980ער: דער אויפשטייג פון עקספערט-סיסטעמען, למשל פאר מעדיצינישע דיאגנאזן.
- די 1990ער: AI ווערט פראקטיש, ווען IBM'ס Deep Blue באזיגט [דעם שאך-מייסטער].
- די 2000ער: AI אין טאג-טעגלעכן לעבן, מיט פונקטן פאר 2005, 2011, 2012, 2013, 2014, 2016, 2017 און 2020.
- 2024 ביז היינט: AI אנטוויקלט זיך וויטער מיט LLMs, אויטאנאמע AI-אגענטן, א.א.וו.

אזוי גיט ChatGPT א ענטפער ווען מען שרייבט דעם פראמפט מיט קלארקייט. ווען מען ווילּ מער פרטים וועגן א געוויסן טיילּ, קען מען וויטער שרייבן למשל "דערצייל מיר מער וועגן נאך 2020", און עס וועט מער דעטאלירן. דער קורס פארגלייכט ביידע ענטפערס: דער איין-שורה'דיגער פראמפט גיט סתם א מאסע אינפארמאציע, דער קלארער גיט א געצילטן ענטפער. מען קען אויך אפשטעלן דאס שרייבן פון ענטפער ווען מען דארף שוין נישט מער.

דעמא: קאנטעקסט (5:23:54)

דער צווייטער עלעמענט איז קאנטעקסט, דאס הייסט צוגעבן פאסיגע הינטערגרונט-אינפארמאציע. מען קאפירט און לייגט אריין אין ChatGPT א ארטיקל פון וועב, און שרייבט ערשט א פראמפט אן קאנטעקסט, נאר "summarize". עס קומט ארויס א טיטל און א סיכום וואס באהאנדלט דעם קלימאט, מענטשליכע ווירקונגען, א.א.וו. דער קורס באמערקט אז א סיכום דארף נישט זיין אזוי לאנג.

אנשטאט דעם שרייבט מען: "סומיר דעם ארטיקל איבער קלימאט-ענדערונג, און הייב ארויס די סיבות און ווירקונגען, אין וויניגער ווי 200 ווערטער", ווייל קיינער ווילּ נישט א סיכום אזוי לאנג. דא איז דער קאנטעקסט געשטעלט:

- עס שטעלט פעסט דעם טעמע, קלימאט-ענדערונג.
- עס פירט אן AI זיך צו קאנצענטרירן אויף סיבות און ווירקונגען.
- עס באגרענעצט די צאל ווערטער, אזוי אז דער סיכום פון דעם ארטיקל איז קורץ.

מען זעט אז דער נייער סיכום איז אסאך קירצער, אונטער 200 ווערטער.

דעמא: ספעציפישקייט (5:26:32)

מען שרייבט "list some programming languages". אזא פראמפט גיט AI נישט קיין ריכטונג וועלכע סארט שפראכן מען דארף. עס גיט א רשימה פון גראדעווילדע שפראכן, טייל פון זיי נישט שייך צו דעם וואס דער באניצער דארף. דער קורס ערקלערט: אויב מען ארבעט נאר אויף וועב-פראגראמירונג, דארף מען נישט שפראכן פאר דאטא-סייענס אדער פאר מאבייל-אפליקאציעס. דערפאר מוז מען פונקטליך זאגן וועלכע שפראכן מען וויל.

מען שרייבט: "list five programming languages commonly used for web development along with their primary use cases". דאס באגרענעצט ערשטנס צו פינף שפראכן. צווייטנס שטעלט עס פעסט נאר וועב-אנטוויקלונג. אין דעם ערשטן ענטפער זענען געווען אויך סקריפטינג, אויטאמאציע, שפיל-אנטוויקלונג און סיסטעם-פראגראמירונג, וואס מען דארף נישט. "צוזאמען מיט זייערע הויפט-אנווענדונגען" מאכט דעם ענטפער מער אינפארמאטיוו און פראקטיש. דער רעזולטאט ברענגט שפראכן ווי JavaScript, יעדע מיט איר אנווענדונג.

דעמא: באגרענעצונגען (5:29:10)

מען שרייבט "explain machine learning". עס איז נישטא קיין אנווייזונג וועגן לענג אדער טאן, און דער ענטפער איז אזוי לאנג ווי ער וויל.

דערנאך שרייבט מען: "שרייב א סיכום פון 100 ווערטער איבער מאשין-לערנען אין א פארמאלן טאן". דאס באגרענעצט די צאל ווערטער צו 100, שטעלט פעסט א פארמאלן אקאדעמישן טאן, און פירט אן AI צו דער ערווארטעטער סטרוקטור.

מען פארגלייכט ביידע ענטפערס. דער ערשטער: "מאשין-לערנען איז א טייל פון קינסטליכער אינטעליגענץ וואס דערמעגלעכט קאמפיוטערס צו לערנען פון דאטא און פארבעסערן זייער ארבעט אן ווערן ספעציפיש פראגראמירט". דער צווייטער: "מאשין-לערנען איז א צווייג פון קינסטליכער אינטעליגענץ וואס דערמעגלעכט סיסטעמען צו לערנען פון דאטא... עס באשטייט פון אלגאריטמען וואס דערקענען מוסטערן, מאכן פאראויסזאגונגען און אויטאמאטיזירן באשלוסן; ML ווערט צעטיילט אין...". דאס איז א ריכטיגער פארמאלער טאן.

דעמא: ביישפילן, און די הויפט-לעקציעס (5:31:12)

נאך איין זאך: ניצן ביישפילן כדי צו באשטימען וואס מען וויל. "Explain Newton's third law" איז צו אפן. עס גיט דריי ביישפילן: שפרינגען, ארויסשיסן א ראקעט, און גיין.

דערנאך שרייבט מען, מיט א קלארן ביישפיל, אז מען וויל די ערקלערונג דורכן ביישפיל פון ארויסשיסן א ראקעט. דער ענטפער: "פאר יעדע אקציע איז דא א גלייכע און פארקערטע רעאקציע". ביי ארויסשיסן א ראקעט ווארפט דער ראקעט-מאטאר ארויס הייסע גאזן אראפ מיט גרויס שנעלקייט. דאס איז די אקציע-קראפט. אזוי קען מען אנפירן ChatGPT מיט ביישפילן.

דער קורס פאסט צוזאם: גוטע פראמפט-אינזשעניריע מאקסימירט די קוואליטעט פון AI-ענטפערס דורך קלארקייט, קאנטעקסט, ספעציפישקייט, באגרענעצונגען און ביישפילן. די הויפט-לעקציעס:

- קלארקייט מאכט זיכער אז AI פארשטייט די בקשה אן צווייטייטשיגקייט.
- קאנטעקסט גיט הינטערגרונט-פרטים כדי צו פארבעסערן די געניקייט.

- ספעציפישקייט פארענגט דעם ענטפער צו דעם וואס דער באניצער מיינט.
- באגרענעצונגען באשטימען סטרוקטור, לענג און סטיל.
- ביישפילן פירן אן AI צו געבן ענטפערס אין דעם געוואונטשענעם פארמאט.

אין דער קומענדיגער אייניהייט וועט מען זען פראקטישע אנווענדונגען פון פראמפט-אינזשעניריע און לערנען פראמפט-מוסטערן (prompt patterns).

אנווענדונגען פון פראמפט-אינזשעניריע (5:33:47)

פראמפט-אינזשעניריע ווערט גענוצט אין אסאך געביטן:

- שאפן אינהאלט: AI שרייבט בלאג-ארטיקלען, אונטערשריפטן פאר סאשעל מידיא, און מעשיות. למשל, מען בעט א AI-שרייבער שרייבן א ארטיקל פון 500 ווערטער איבער דער ווירקונג פון AI, און מען שטעלט פעסט די סטרוקטור, דעם טאן, און די הויפט-פונקטן וואס דארפן אריינקומען. א גוט-סטרוקטורירטער פראמפט מאכט זיכער אז דער רעזולטאט איז שייך, צוזאמענהענגיג, און פליסט לאגיש.
- הילף מיט קאוד: AI-געהילפן שאפן שטיקלעך קאוד און העלפן געפינען טעותים. למשל: "שרייב א Python-פונקציע וואס קוקט אויב א געגעבענער string איז א palindrome". AI שרייבט א פונקציע לויט Python'ס בעסטע פראקטיקן. אז מען לייגט צו באגרענעצונגען ווי "אפטימיזיר פאר שנעלקייט" און "ניץ נישט קיין recursion", ווערט דער רעזולטאט נאך מער עפעקטיוו.
- קונה-סערוויס: AI-טשאטבאטס ענטפערן פראגעס פון קונים מיט ווייניגער מענטשליכע אריינמישונג. א פראמפט ווי "גיב א שריט-ביי-שריט אנווייזונג צו פארריכטן Wi-Fi פראבלעמען אויף Windows 11" גיט א בעסערן ענטפער ווי א שוואכע פראגע. צו די בעסטע פראקטיקן געהערט צולייגן באגרענעצונגען ווי "ענטפער אונטער 200 ווערטער און ניץ פשוט'ע שפראך", כדי עס זאל זיין קלאר און קורץ.
- דאטא-אנאליז: AI-מאדעלן מאכן סיכומים פון באריכטן און ציען ארויס אינפארמאציע פון גרויסע דאטא-סעטס. למשל, א פינאנץ-אנאליסט שרייבט: "סומיר די הויפט-טרענדס פון איינקונפט אין דער פירמע'ס Q4 באריכט אין דריי פונקטן". דאס ציט ארויס די וויכטיגע אינפארמאציע עפעקטיוו, בשעת דער סיכום בלייבט קורץ און מען קען דערויף האנדלען.
- חינוך און לערנען: AI-טוטארן ערקלערן באגריפן אין פרטים. למשל: "ערקלער דעם Pythagorean theorem מיט א ביישפיל פונעם לעבן", כדי AI זאל געבן אן אינטואיטיוו ערקלערונג מיט פראקטישער אנווענדונג. מען קען דעם ענטפער נאך פארבעסערן אז מען לייגט צו: "ניץ א משל אין פארעם פון א מעשה מיט א דרייעק מיט א גראדן ווינקל".

פראמפט-מוסטערן און געווייניטלעכע פראמפט-טעותים (5:36:21)

פיר הויפט-סארטן פראמפט-מוסטערן (5:36:21)

אין דעם דעמא-טייל ערקלערט דער קורס די פארשידענע מוסטערן (patterns) אין פראמפט-אינזשעניריע. פראמפט-אינזשעניריע נעמט אריין פארשידענע מוסטערן וואס העלפן אויפבויען דעם שמועס מיט'ן AI אויף אן עפעקטיוו אופן. עס ווערן אויסגערעכנט פיר הויפט-סארטן:

- אינסטרוקציע-באזירטע פראמפטס (instruction-based prompts)
- ביישפיל-געטריבענע פראמפטס (example-driven prompts)

- "אויספילן די ליידיגע ערטער" פראמפטס (fill-in-the-blanks prompts)
- שריט-בי-שריט פראמפטס (step-by-step prompts)

פאר יעדן סארט ווערן געוויזן צוויי-דריי ביישפילן אין ChatGPT.

אינסטרוקציע-באזירטע פראמפטס (5:36:52)

אינסטרוקציע-באזירטע פראמפטס זענען דירעקטע באפעלן וואס ברענגען ארויס ספעציפישע ענטפערס. זיי זענען זייער עפעקטיוו פאר סטרוקטורירטע רעזולטאטן, ווייל זיי זיכערן קלארקייט און פינקטלעכקייט. דער עיקר איז צו זאגן גאנץ בפירוש וואס מען וויל אז דער AI זאל טון, למשל צוזאמענפאסן, ערקלערן אדער פארמאטירן אינהאלט.

ערשטער ביישפיל: מען שרייבט "Explain quantum mechanics in simple terms". ווייל מען האט צוגעלייגט "in simple terms", גיט דער AI אן ערקלערונג פאר אנהייבערס אן קיין טעכנישע אויסדרוקן. מען זעט אז דער ענטפער ברענגט א דעפיניציע, עטליכע הויפט-געדאנקען, און צום סוף א צוזאמענפאסונג.

צווייטער ביישפיל: מען שרייבט "Summarize this in one paragraph", מיינענדיג דעם פריערדיגן ענטפער. דער AI ציט ארויס די הויפט-פונקטן און גיט א קורצע ווערסיע. מען זעט אז דער ענטפער איבער quantum mechanics קומט יעצט טאקע אין איין פאראגראף. דאס איז אן אינסטרוקציע-באזירטער פראמפט ווייל מען האט געגעבן די אינסטרוקציע "summarize", פונקט ווי פריער "in simple terms".

דריטער ביישפיל: "Write an email to my manager requesting for two days" (בעטן צוויי טעג פריי). דער AI שטעלט צוזאם א פראפעסיאנעלע און העפליכע בקשה: עס הייבט זיך אן מיט א באגריסונג צום מענעדזשער מיט אן ארט פאר זיין נאמען, א וווּנטש אז ער זאל זיך גוט פילן, א בקשה פאר די טעג צוליב א געוויסער סיבה, און א צוזאג אז אלע ארבעטן וועלן זיין אין ארדענונג פאר'ן ווייל-ניין, און אז מען וועט זיך קאארדינירן מיט'ן טיעם אויב נויטיג. דער שטייגער איז זייער גלייך און פאסט גוט פאר אויפגאבעס וואו מען דארף קורצע, קלארע אינסטרוקציעס.

ביישפיל-געטריבענע פראמפטס (5:39:30)

ביישפיל-געטריבענע פראמפטס העלפן דעם AI פארשטיין דעם ערווארטעטן פארמאט און טאן, דורך געבן איינס אדער מער ביישפילן איידער מען בעט דעם ענטפער. די טעכניק איז זייער נוצלעך פאר שאפן קרעאטיוון אינהאלט, איבערזעצונגען אדער סטיל-ענדערונגען. דער AI לערנט זיך פונעם מוסטער וואס די ביישפילן שטעלן אוועק און גייט נאך די זעלבע סטרוקטור.

ערשטער ביישפיל: "Generate a poem about winter in the style of Robert Frost". ChatGPT נאכמאכט דעם סטיל פון Frost מיט טעמעס פון דער נאטור און לעבעדיגע בילדער. דא האט ChatGPT געגעבן צוויי ענטפערס און מען קען אויסקלייבן איינעם. אין די שורות זעט מען פראסט וואס קריכט ארויף אויף'ן וועג דורכ'ן וואלד, א שטילקייט וואס הענגט אויף די צווייגן, און ווינטער וואס בייגט די סאסנע-ביימער. מען קלייבט "I prefer this response", און מען זעט אז דער סטיל פון Frost ווערט טאקע נאכגעמאכט.

צווייטער ביישפיל: איבערזעצונג. מען שרייבט "Translate the following: Hello, how are you" און אנשטאט צו שרייבן "Spanish" שרייבט מען בלויז "Hola". ChatGPT כאפט פון זיך אליין אז עס דארף זיין שפאניש, און זעצט איבער "Hello, how are you" אויף שפאניש. נאכדעם גייט ChatGPT נאך דעם זעלבן

פארמאט ביי ווייטערדיגע איבערזעצונגען: וועו מען שרייבט "Translate: My name is Sam", קומט "Me llamo Sam" אויף שפאניש, הגם מען האט דאס מאל נישט אנגעגעבן קיין שפראך אדער קיין סימן.

דריטער ביישפיל: "Rewrite this sentence like a Shakespearean monologue: The sun is shining brightly today". דער ענטפער: דער גאלדענער באלעם פלאקערט מיט שטראלנדיגער מאכט און ווארפט א הימלישן גלאנץ אויף דער ערד. ChatGPT האט פארוואנדלט דעם זאץ אין אלט-ענגלישער פראזע.

דאס מוסטער איז עפעקטיוו וועו מען דארף היטן אויף קאנסיסטענץ און סטיל, כדי דער AI זאל ארויסגעבן ענטפערס אין א פאראויס באשטימטן פארמאט. עצה: אויב מען לאדט ארויף עפעס, קען מען שרייבן צו ChatGPT "איך וויל אוא סארט רעזולטאט", און ChatGPT וועט נאכגיין די זעלבע סטרוקטור.

"אויספילן די ליידיגע ערטער" פראמפטס (5:43:42)

די פראמפטס מוטיגן דעם AI צו פארענדיגן א זאץ אדער א פאראגראף לויט די אנצוהערעניש פונעם קאנטעקסט. מען נוצט דאס אפט פאר פריי קרעאטיוו שרייבן, טריוויע-פראגן אדער ווייסן-איבערחזרן. זיי לאזן דעם AI ארויסגעבן פלעקסיבלע אבער סטרוקטורירטע ענטפערס, און פירן זיין רעזולטאט אן צו זיין צו שטרענג באגרענעצט.

ערשטער פרואוו: "My name is _". דער AI האט עס אויפגענומען אלס א פראגע ווי אזוי מען זאגט עס אין אן אנדער שפראך, און געענטפערט מיט א פארמאט "איך הייס [נאמען]".

צווייטער ביישפיל: "A black hole is formed when _". ChatGPT פארענדיגט דעם זאץ מיט א וויסנשאפטלעך פינקטלעכער דערקלערונג: א שווארצע לאך ווערט געשאפן וועו א ריזיגער שטערן קומט צום סוף פון זיין לעבן און פאלט איין אונטער זיין אייגענער גראוויטאציע.

דריטער ביישפיל: "The key to success in life is _". דא פילט ChatGPT אויס מיט פארשידענע פערספעקטיוון, ווי צום ביישפיל אויסדויער און המשך'דיגע לערנען: "דער שליסל צו הצלחה אין לעבן איז א צוזאמשטעל פון אויסדויער, המשך'דיגע לערנען, צופאסונג-פעאיגקייט" און נאך אפציעס.

פערטער ביישפיל, מיט מער ווי איין ליידיג ארט: "If I could travel anywhere in the world, I would go to _ because _". ChatGPT קלייבט אויס א ציל און גיט א לאגישע סיבה: יאפאן, ווייל עס פארבינדט אלטע טראדיציעס מיט מאדערנער אינאוואציע, האט אטעם-צונעמענדיגע לאנדשאפטן און א רייכע קולטורעלע ירושה, פון רואיגע אלטע געביידעס און בלומען-ביימער ביז פוטוריסטישע שטעט און א באזונדערע קיך. דאס מוסטער איז באזונדערס נוצלעך כדי צו פארנעמען באניצער מיט אינטעראקטיווע אויפגאבעס, און פאסט גוט פאר קוויזן, לערן-מאטריאל און ענטפערס וואס ברענגען צום טראכטן.

שריט-ביי-שריט פראמפטס (5:47:15)

שריט-ביי-שריט פראמפטס צעטיילן קאמפליצירטע טעמעס אויף עטליכע שריט, ווי א פארמל אדער אן אלגאריטם, כדי זיי זאלן זיין גרינגער צו פארשטיין. מען נוצט זיי אפט אין טעכנישע ערקלערונגען, באשרייבונגען פון פראצעסן און פראבלעם-לייזונג. וועו מען מוטיגט דעם AI צו ערקלערן אין א סדר, ווערט עס קלארער און דער ענטפער גייט מיט א לאגישן פלוס, אזוי אז עס איז גרינג צו פארשטיין סיי וואס, צי עס איז מאטעמאטיק פון 12טן קלאס צי דער Pythagoras טעארעם.

ערשטער ביישפיל: "Describe the water cycle in four sequential steps". ווייל מען האט צוגעלייגט "in four sequential steps" קומען טאקע פיר שריט: אויסדאמפונג (evaporation), קאנדענסאציע

(condensation), אראפפאל (precipitation) און צוזאמזאמלונג (collection), מיט אן ערקלערונג פאר יעדן שריט.

צווייטער ביישפיל: "Explain how to bake a cake step by step". מען זעט ערשט די אינגרעדיענטן און נאכדעם די שריט. אויב מען וויל בלויז די שריט, קען מען צולייגן "I need only steps".

דריטער ביישפיל: "Solve this math problem step by step: 2x + 5 = 15". ChatGPT צעלייגט עס: ערשטער שריט, אראפנעמען 5 פון ביידע זייטן; דערנאך טיילן ביידע זייטן אויף 2; דער ענדגילטיגער ענטפער: x = 5. די מעטאד איז זייער עפעקטיוו פאר ערקלערן סטרוקטורירטע פראצעסן און פארבעסערט לערן-אינהאלט פון AI פאר לערן-אפליקאציעס.

סיכום: יעדער פון די פראמפטס האט זיין אייגענעם צוועק און פארבעסערט דעם AI-רעזולטאט לויט דעם באדארף. צי מען דארף א סטרוקטורירטן ענטפער, קאנסיסטענטן סטיל, פרייַע קרעאטיוויטעט אדער א דעטאלירטע ערקלערונג, מאכט דאס ריכטיגע מוסטער די ענטפערס פינקטלעכער, נוצלעכער און מער פארכאפנדיג.

גענוייַנטלעכע פראמפט-טעותים: צו אומקלאר (5:50:55)

דער נעקסטער מאדול באהאנדלט גענוייַנטלעכע טעותים ביים שרייבן פראמפטס און ווי אזוי זיי צו פארריעכטן. די ענטפערס פון ChatGPT אדער AI הענגען שטארק אפ ווי אזוי מען בויט דעם פראמפט, און אפילו דערפארענע באניצער מאכן טעותים וואס פירן צו אומקלארע, אומפולשטענדיגע אדער צעמישטע רעזולטאטן.

ערשטער טעות: צו אומקלאר (too vague). אן אומקלארער פראמפט פעלט גענוג פרטים, און עס ווערט שוואר פאר'ן AI צו גענבן א פינקטלעכן אדער נוצלעכן ענטפער; אן קלארע אינסטרוקציעס קען דער AI ארויסגעבן אן אלגעמיינעם ענטפער וואס פאסט נישט צום באדארף. ביישפיל: "Tell me about the economy". דאס איז זייער ברייט און אן ריכטונג: אינדיע'ס עקאנאמיע? אמעריקע'ס? עקאנאמיע איז א ריזיגע טעמע וואס נעמט אריין GDP, אינפלאציע, באשעפטיגונג, וועלט-האנדל און נאך, און דער AI וויסט נישט אויף וועלכן אַספעקט מען וויל זיך קאנצענטרירן. דער ענטפער: א דעפיניציע פון עקאנאמיע אלס א סיסטעם פון פראדוקציע, סארטן עקאנאמיעס (מארקעט-עקאנאמיע, באפעל-עקאנאמיע, געמישטע עקאנאמיע), און עקאנאמישע אנווייזערס ווי GDP, אינפלאציע-ראטע, באשעפטיגונג-ראטע און דער עקאנאמישער ציקל. אבער וואס אויב מען דארף דאס גארנישט?

די פארריכטונג: "Explain the impact of inflation on consumer spending in the US in 2025". דער ענטפער רעדט יעצט ספעציפיש וועגן ווי אינפלאציע אין 2025 האט שטארק באאיינפלוסט די קויף-מוסטערן פון קאנסומערס אין די פאראייניגטע שטאטן, מיט מקורות פון וועב. פארוואס איז דער פראמפט בעסער? ער שמאלט איין דעם פאקוס אויף אינפלאציע אנשטאט דער גאנצער עקאנאמיע, ער באשטימט קאנסומער-אויסגאבן און שניידט אפ נישט-שייכותדיגע פרטים, און ער באשטימט די געגנט (US) און די צייט (2025). וואס ספעציפישער די פראגע, אלס שייכותדיגער און פראקטישער דער ענטפער.

צווייטער טעות: איבערלאדן מיט אינפארמאציע (5:54:07)

דער צווייטער טעות איז צו פיל פרטים אין איין בקשה, וואס קען איבערשוועמען דעם AI און פירן צו אן אומפאקוסירטן אדער אומפולשטענדיגן ענטפער; דער AI קען געבן פריאריטעט צו געוויסע פרטים און איגנאריר אנדערע. ביישפיל: "Explain how blockchain works, including its history, advantages, disadvantages and comparison with centralized databases". דא בעט מען צופיל אויף איינמאל:

היסטאריע, מעכאניזם, מעלות, חסרונות און א פארגלייך. דער פראבלעם: דער AI קען געבן אויבנאויפיגע ענטפערס פאר יעדער טעמע אנשטאט א טיפע ערקלערונג; דער ענטפער קען זיין צו לאנג אדער צעמישט; וויכטיגע פונקטן קענען אויסגעלאזט ווערן; און עס ווערט שוואך צוזאמגעפאסט צוליב באגרענעצטן ארט. מען זעט טאקע אז יעדער טעמע באקומט בלויז איין פונקט.

די פארראכטונג: "Explain how blockchain works in simple terms with one real-world use case". דער ענטפער ערקלערט יעצט ווי blockchain ארבעט שריט ביי שריט, וועלכן פראבלעם עס לייזט, און פארוואס עס איז נוצלעך. דאס איז פיל בעסער ווייל דער פאקוס איז בלויז אויף ווי blockchain ארבעט אנשטאט אויף אסאך אַספעקטן, די בקשה איז עס צו האלטן פשוט כדי צו זיכערן קלארקייט, און א פראקטישער ביישפיל מאכט דעם ענטפער מער פארשטענדלעך.

פראמפט-טעותים און פארגעשריטענע טעכניקן (5:57:17)

א צווייטער טעות: צו ברייטע פראגעס, און ווי צו צעטיילן (5:57:17)

דער קורס ענדיגט דא ארויס דעם פריערדיגן ביישפיל וועגן blockchain. מען זעט אז וועז מען האט געגעבן א קאנקרעטן פראבלעם, למשל אז די עסנווארג-supply chain ליידט פון ניט-עפנטלעכקייט (transparency) און פון א פאמעלעכן נאכשפירן פון פארפעסטע פראדוקטן, איז די ערקלערונג געווארן מער פראקטיש און מער שייך.

עס ווערט ערקלערט אז אויב מען דארף מער פרטים, קען מען צעטיילן די פראגע אין באזונדערע פראמפטס, אנשטאט אלעס אריינצושטופן אין איין פראגע. למשל:

- "וואס איז די היסטאריע פון blockchain?"
- "וואס זענען די מעלות און חסרונות פון blockchain?" (באזונדער)
- "ווי פארגלייכט זיך blockchain מיט א צענטראליזירטער דאטא-באזע אין שייכות צו פעסטקייט?"

אויף אזא אופן באקומט מען א קורצע, קלארע ערקלערונג אויף יעדן פונקט באזונדער.

דריטער טעות: מען באשטימט ניט די פארמאט פונעם ענטפער (5:57:48)

דער דריטער טעות הייסט "מען אינגנארירט די פארמאט פונעם ארויסגאנג". דער פראבלעם: אויב מען זאגט ניט ווי אזוי דער ענטפער זאל זיין אויסגעשטעלט, קען דער AI אים צוריקגעבן אין א פארמאט וואס העלפט ניט. דאס מאכט עס שווערער צו לייענען אדער ניצן די אינפארמאציע.

א ביישפיל פון אן ניט-סטרוקטורירטן פראמפט: "גיב די מעלות פון געניטונג (exercise)". מען וואלט געקענט ערווארטן אז ChatGPT וועט ארויסגעבן א לאנגן פאראגראף אנשטאט א קלארע רשימה. אין דער לעקציע ווערט עס גענאפן, און ס'קומט ארויס א טייל רשימה, אבער די הויפט-פונקטן קענען זיך פארלירן אינעם טעקסט מיט די emojis און אזוי ווייטער, אנשטאט ארויסצושטעכן.

דער תיקון: מען שרייבט "רעכן אויס די פינף הויפט-מעלות פון געניטונג אין bullet points". פארוואס איז דער פראמפט בעסער?

- ער פארזיכערט א קורצן, סטרוקטורירטן ענטפער.
- bullet points פארבעסערן די לייענבארקייט און קלארקייט.

- דאס באגרעניצן צו פינף מעלות מאכט דעם ענטפער מער פאקוסירט.

דער ענטפער קומט ארויס פשוט און קורץ, "simple and crisp", בעסער ווי פריער. מען קען אויך בעטן אנדערע פארמאטן, למשל:

- "שאף א טאבעלע וואס פארגלייכט aerobic און anaerobic ענ׳יטונג לויט מעלות און ביישפילן".
- "ערקלער ווי אזוי אנצוהייבן א ענ׳יטונג-רוטין אין פינף פשוטע טריט".
- "פאס צוזאם די מעלות פון כח-טריינירונג (strength training) אין צוויי קורצע פאראגראפן".

פערטער טעות: מען פאליר ניט די פראמפטס (6:00:38)

א פערטער אפטער טעות איז אז מען פארבעסערט ניט דעם פראמפט. איין פרואוו גיט ניט שטענדיג דעם בעסטן ענטפער. אינהאלט פון ChatGPT אדער אן אנדער AI קען זיין צו אלגעמיין, צו דעטאלירט, אדער אביסל ניט צום ענין. דאס פאלירן פונעם פראמפט פארבעסערט די ענויקייט פונעם רעזולטאט.

ביישפיל: "שאף א sales pitch". דאס איז צו ברייט און פעלט אינהאלט. ChatGPT ווייסט ניט פאר וועלכן פראדוקט אדער סערוויס דער pitch איז, דעריבער קען דער ענטפער זיין צו אלגעמיין צו העלפן, און ער פאסט אפשר ניט צום ענוואונטשענעם טאן אדער עולם.

אנשטאט דעם שרייבט מען: "שאף א sales pitch פון 100 וועדטער פאר אן AI-ענטריבענער fitness-אפליקאציע, וואס פאקוסירט אויף פערזענלעכקייט (personalization) און עפעקטיווקייט". פארוואס איז עס בעסער?

- דער פראדוקט איז באשטימט: אן AI-ענטריבענע fitness-אפליקאציע.
- די לענג איז באשטימט, 100 וועדטער, כדי עס זאל בלייבן קורץ.
- עס הייבט ארויס די הויפט-פארקויף-פונקטן: personalization און עפעקטיווקייט.

מען לאזט עס לויפן. אויב דער ענטפער איז נאך ניט פונקט וואס מען דארף, קען מען עס ווייטער פאלירן, למשל "מאך דעם pitch מער איבערצייגנדיק דורך צולייגן א ספעציעלע אפערטע", אדער "שרייב איבער דעם pitch מיט א מער ציענדיקן, שמועסדיקן טאן". נאכן איבערשרייבן זעט מען א נייעם ענטפער, וואס הייבט זיך אן בערך מיט די וועדטער "נישט צופרידן מיט אייערע ענ׳יטונגען? אייער פערזענלעכער AI fitness-קאוטש וואס פאסט צו די ענ׳יטונגען...", און עס איז בעסער ווי פריער. מען קען אזוי ווייטער פאלירן לויט דעם וואס מען דארף.

די סוף-לעקצ׳עס פון דעם טייל: זיין ספעציפיש, בלייבן פאקוסירט, באשטימען די פארמאט, און פאלירן און איבערחזרן. וועו מען אויסמיידט די דאזיגע טעותים און ניצט די תיקונים, פארבעסערט מען די ענויקייט, די קלארקייט און די נוצבארקייט פונעם AI-אינהאלט.

פארגעשריטענע טעכניקן: Zero-shot learning (6:03:45)

דער קורס גייט ווייטער צו פארגעשריטענע טעכניקן פון prompt engineering: zero-shot, few-shot און chain of thought prompting. עס וועדט ערקלערט אז ביים ארבעטן מיט AI שפראך-מאדעלן האט די סטרוקטור פונעם פראמפט א שטארקע השפעה אויף דער קוואליטעט פונעם ענטפער. יעדע טעכניק האט אן אייגענעם וועג צו פירן דעם AI צו שייכותדיקע און ענויע ענטפערס.

Zero-shot learning מיינט אז מען בעט דעם מאדעל צו געבן א ענטפער אן קיין פריערדיגע ביישפילן אדער אנווייזונגען. דער AI אדער ChatGPT פארלאזט זיך אינגאנצן אויף זיין פאר-טריינירטע וויסן, נאר אויפן יסוד פונעם פראמפט אליין. מען ניצט עס וועו מען ערווארט אז דער מאדעל זאל קענען אלגעמיינערן און ענטפערן אן א ספעציפישן ביישפיל.

ביישפיל: דער פראמפט "ערקלער די וויכטיגקייט פון cybersecurity אין ביזנעס". דער ענטפער הייבט זיך אן בערך אזוי: "cybersecurity איז קריטיש פאר ביזנעסער ווייל עס העלפט...".

ווי ארבעט עס? zero-shot learning איז ווי מען וואלט געפרעגט א מענטש א פראגע אויף א טעמע וואס ער האט געלייענט דערוועגן, אבער אן קיין נאכאנאנדיקע אנווייזונגען. דער AI מוז אליין ארויסנעמען דעם ריכטיגן ענטפער פון זיין טריינירונג-דאטא און וויסן-באזע.

מעלות:

- עס איז זייער שנעל און עפעקטיוו: דער AI קען גלייך ענטפערן אן נאך קאנטעקסט.
- עס ארבעט גוט פאר אלגעמיינע וויסן-פראגעס: דער AI גיט א פארנינפטיגן ענטפער אויפן יסוד פון זיין פאר-טריינירטן מאדעל.
- עס איז אידעאל פאר אפענע פראגעס אויף ברייטע טעמעס, ווי די cybersecurity-ביישפיל, וואו מען דארף ניט קיין דעטאלירטע ערקלערונג.

שווערigקייטן:

- דער AI קען געבן אלגעמיינע אדער וויניגער געוויע ענטפערס, ווייל ער האט ניט קיין ספעציפישע אנווייזונגען.
- קאמפליצירטע אויפגאבעס קענען פארלאנגען מער קאנטעקסט פאר בעסערע געוויקייט.

נאך א zero-shot ביישפיל: "באשרייב blockchain-טעכנאלאגיע און איר השפעה אויף פינאנציעלע זיכערהייט". דער מאדעל גיט א ענטפער אן קיין פריערדיגע ביישפילן, נאר מיט זיין אייגן וויסן.

(6:07:50) Few-shot learning

Few-shot learning פארבעסערט דעם AI'ס ארבעט דורך געבן אים א פאר ביישפילן פאר מען בעט אים צו ענטפערן, דערפאר הייסט עס "few-shot". דאס העלפט דעם AI דערקענען א מוסטער און ארויסגעבן מער שייכותדיקע, סטרוקטורירטע און געוויע ענטפערס. עס איז זייער נוצלעך וועו מען וויל אז דער AI זאל נאכגיין א געוויסן טאן, סטרוקטור אדער ערווארטעטע ענטפער-פארעם, למשל אויב מען דארף א ענטפער אין א ליד-סטיל.

אין דער דעמא שרייבט מען אין איין פראמפט צוויי ביישפילן:

- ביישפיל 1: א שטארקער password אנטהאלט גרויסע אותיות, ציפערן און סימבאלן.
- ביישפיל two-factor authentication :2 לייגט צו א נאך שיכט זיכערהייט.

דערנאך: "יעצט ערקלער וואס איז די וויכטיגקייט פון אפדעיטן ווייכווארג (software)", באגרעניצט צו קנאפע 200 ווערטער.

ווי ארבעט עס? few-shot learning איז ענליך צו ווי מענטשן לערנען פון ביישפילן און משלים. וועו מען גיט אים פריער ביישפילן, פארשטייט דער AI וואס מען ערווארט. דער ענטפער: אפדעיטן ווייכווארג איז קריטיש;

למשל, אויב א operating system אדער web browser האט א זיכערהייט-לאך, קענען אטאקירער עס ניצן אריינצושטעקן שעדלעכן קאוד אדער דורכפירן ransomware-אטאקעס. און גלייך ווי א שטארקער password און two-factor authentication לייגן צו שיכטן זיכערהייט, אזוי טוען אויך רעגולערע software-אפדעיטס. מען זעט אז דער ענטפער פאלגט נאך דעם מוסטער פון די ביישפילן.

מעלות:

- מער גענויע און קאנטעקסטועלע ענטפערס: דער AI לערנט פון די ביישפילן און פאלגט דעם מוסטער.
- בעסער פאר ספעציפישע ניצונגען, דא: ווייכווארג און זיכערהייט.
- עס פארבעסערט דעם AI'ס פארשטאנד; ער אלגעמיינערט בעסער ווען מען גיט אים סטרוקטורירטע פראמפטס.

שווערigקייטן:

- מען דארף גוט אויסקלייבן די ביישפילן, כדי דער AI זאל פארשטיין דעם געוואונטשענעם רעזולטאט; דערפאר איז prompt engineering אזוי וויכטיג.
- מער ביישפילן קענען פארלענגערן די צייט ביז דער ענטפער ווערט ארויסגעגעבן. אבער היינט איז דער מאדעל זייער קלוג און עס נעמט ניט פיל לענגער; צוויי-דריי ביישפילן זענען גענוג.

(6:13:01) Chain of thought prompting

Chain of thought prompting העלפט דעם AI צעברעכן א קאמפליצירטע אויפגאבע אין לאגישע טריט. די מעטאד איז באזונדערס עפעקטיוו פאר זאכן וואס פארלאנגען טראכטן און שלוסן, וואו דער AI דארף ערקלערן א פראצעס אדער לייזן א פראבלעם שריט נאך שריט.

ביישפיל: "ערקלער דעם פראצעס פון photosynthesis אין דריי טריט" (אן דעטאלירטע טריט, נאר די טריט אליין). דער ענטפער ערקלערט קודם קורץ וואס photosynthesis איז, און דערנאך די טריט.

ווי ארבעט עס? אנשטאט צו בעטן א דירעקטן ענטפער, מוטיגט chain of thought prompting דעם AI צו דורכטראכטן דעם פראבלעם אדער ערקלערונג אין עטלעכע שטאפלען. דאס פארבעסערט די קלארקייט און העלפט ארויסגעבן גוט-סטרוקטורירטע ענטפערס. מען זעט: ערשטער שריט, צווייטער שריט, דריטער שריט, און עס איז זייער גרינג פאר אונז צו פארשטיין.

נאך א ביישפיל: "באשרייב ווי אזוי ען עלעקטרישער (capacitorקאנדענסאטאר) האלט אפ ענערגיע". דער ענטפער: א קורצע דעפיניציע, און דערנאך ערשטער, צווייטער און דריטער שריט. אויב ער וואלט געגעבן לאנגע פאראגראפן, וואלט עס שווער געווען צו פארשטיין; דערפאר דארף מען וויסן ווי צו ניצן chain of thought prompting.

מעלות:

- עס פארשטארקט לאגיש טראכטן, מיט שריט-נאך-שריט ערקלערונגען.
- עס איז אידעאל פאר קאמפליצירטע טעמעס: ערקלערן פראצעסן, לייזן מאטעמאטישע פראבלעמען אדער שווערע באגריפן.
- עס מוטיגט דעם AI צו טראכטן סטרוקטורירט, אין א לאגישער סדר, אנשטאט איין ענטפער.

שווערigקייטן:

- מאל זענען די ענטפערס לאנג און נעמען צייט צו פראצעסירן.
- דער AI דארף קלארע פראמפטס כדי צו צעברעכן די אינפארמאציע ריכטיג.

סך-הכל און סיום פונעם ChatGPT-טייל (6:16:21)

יעדע טעכניק דינט א צווייטן צוועק:

- zero-shot: וועו דער AI מוז שנעל ענטפערן אן קיין אנווייזונג.
- few-shot: וועו דער AI דארף נאכגיין א געוויסן מוסטער אדער סטרוקטור.
- chain of thought: וועו דער AI מוז דורכטראכטן א פראבלעם אדער ערקלערן א פראצעס אין טריט.

וועו מען פארשטייט און ניצט זיי ריכטיג, קענען באניצער אפטימירן דעם AI פאר פארשידענע אויפגאבעס: פון ענטפערן אלגעמיינע פראגעס ביז שאפן סטרוקטורירטן אינהאלט און לייזן קאמפליצירטע פראבלעמען. דער קורס באטאנט אז prompt engineering איז א וויכטיגע פעאיקייט: וועו מען פארשטייט די באשטאנדטיילן, אויסמיידט די אפטע טעותים און ניצט סטרוקטורירטע טעכניקן, פארבעסערט מען די געוויקייט און עפעקטיווקייט פון AI-אפליקאציעס. מען דארף ווייטער פראקטיצירן און פאלירן די פראמפטס.

דערמיט ענדיגט זיך דער ChatGPT prompt engineering טייל, מיט א רוף צו אריינשטעלן די פעאיקייטן אין די טאג-טעגלעכע ארבעט און עקספערימענטירן מיט פראמפטס.

א נייער טייל: context engineering (6:17:57)

דערנאך הייבט זיך אן א נייער טייל: וואס איז context engineering. עס ווערט געגעבן א משל פון א "picky" (איבערגעקליבענעם) קוכער: שטעלט אייך פאר אז מען דינגט א קוכער און זאגט אים נאר "מאך מיר וועטשערע", אן א רשימה פון אינגרידיענטן, אן קיין דיעטע-באמערקונגען, און אן צו זאגן וויפיל געסט עס וועלן זיין.

קאנטעקסט-אינזשענירינג קעגן "וייב-קאודינג" (6:18:29)

דער קאך-משל און די דעפיניציע פון קאנטעקסט (6:18:29)

דער קורס פירט ווייטער דעם משל פון א קאך. אן קאנטעקסט איז דער רעזולטאט פון דער מאלצייט ווי א רולעטקע, א פולע שפיל אויפן גליק. קאנטעקסט-אינזשענירינג גיט אבער דעם קאך אלעס וואס ער דארף: די אינגרעדיענטן וואס ליגן אין שפייזקאמער, די נאטיצן וועגן דיעטע (קיין נ'ס, פאסיג פאר א וועגאן), וואס מען האט שוין געגעסן ביי פריערדיגע מאלצייטן כדי מען זאל זיך נישט איבערחזר'ן, און דעם סטיל ווי מען האט ליב אויסצושטעלן דאס עסן אויפן טעלער.

אין AI-שפראך ערקלערט מען עס אזוי: קאנטעקסט איז גלייך צו כללים, פלוס דאטא, זכרון, געצייג און דער געוואונטשענער רעזולטאט. דאס איז א צוגעגרייטע סביבה וואס לאזט א גרויסן שפראך-מאדעל טראכטן און נישט טרעפן.

קאנטעקסט-אינזשענירינג אין AI באדייט אויסצוארדענען און צו פירן אלע עלעמענטן וואס א געוויסע ארבעט פאדערט: כללים, דאטא, זכרון, געצייג און דער רעזולטאט. וועו דער AI באקומט די גאנצע נויטיגע הינטערגרונט-אינפארמאציע, קען ער מאכן קלארע און פארלעסליכע באשלוסן אנשטאט צו טרעפן, און ער

ארבעט גוט אויך אין קאמפליצירטע מצבים. בקיצור, עס איז די קונסט צו געבן דעם מאדעל דעם גאנצן קאנטעקסט, אזוי אז די ארבעט זאל זיין מעגליך צו לייזן.

פארוואס "וייב-קאודינג" איז דורכגעפאלן (6:19:32)

פארוואס האט מען בכלל געדארפט דעם טערמין? ווייל מען האט פריער פרובירט פונקט פארקערט. דער קורס גייט צוריק צום אנהייב 2024. די AI-וועלט האט דעמאלט געבליט, יעדע וואך זענען ארויס נייע געצייג, און פראגראמירער אומעטום זענען געווען פארכאפט אין עפעס וואס הייסט "vibe coding". דער געדאנק איז געווען צו זאגן דעם AI-געהילף אויף אן אומקלארן אופן וואס מען וויל, למשל "בוי א to-do אפליקאציע", "מאך א לאנדינג-בלאט פאר מיין סטארטאפ", "גיב מיר א טשעטבאט וואס ענטפערט ווי Shakespeare", און אט האט מען שוין דעם קאוד.

עס האט אויסגעזען ווי כישוף. מען האט נישט געדארפט קיין צוגרייטונג, נישט טראכטן און נישט פלאנירן, נאר "vibes". דאס איז געווען גאר שפאסיג ביי hackathons, סוף-וואך פראיעקטן, שנעלע פראטאטיפן, און פאר דעם "וואו" מאמענט פאר פריינט אדער מיטארבעטער. עס איז אויך געווען ממש סוחר-מאכנדיג: דער דאפאמין-שטויס ווען מען זעט ווי דער AI שרייבט גאנצע שטיקער קאוד אן שום מי. דער קלאפ איז אבער געקומען ווען מען האט געפרובירט ארויסצולאזן די פראיעקטן אדער זיי צו ניצן אין עכטע פראדוקציע.

די פראבלעמען מיט וייב-קאודינג (6:20:36)

וייב-קאודינג האט אויסגעזען ווי מען נארט אפ די סיסטעם, אבער למעשה האט מען מערסטנס אפגענארט זיך אליין. דער קורס רעכנט אויס די פראבלעמען:

- אויסגעטראכטע API'ס: דער AI האט מיט זיכערקייט גענוצט פונקציעס, ביבליאטעקן אדער endpoints וואס עקזיסטירן גארנישט. למשל א פונקציע צו ברענגען דאטא וואס איז געווען ממש אויסגעטראכט.
- קיין מעגליכקייט צו וואקסן (scalability): דער AI האט נישט אויסגעפלאנט דעם קאוד פאר שפעטערדיגן באנוץ. קיין מאדולארקייט, א שוואכע סטרוקטור פון די פיילס, און גארנישט פון קאמענטארן אדער דאקומענטאציע.
- שוואכע טעסטן אדער בכלל נישט: רוב טעסטן וואס דער AI האט געשריבן האבן אדער נישט געפאסט צו דער לאגיק פונעם קאוד, אדער איבערגעהיפט ראנד-פעלער (edge cases), אדער זענען בכלל נישט געווען.

דערפאר, ווען דער קאוד האט געדארפט וואקסן אדער זיך אנטוויקלען, איז ער פשוט איינגעפאלן.

וואס די דאטא ווייזט (6:21:08)

דאס איז נישט געווען בלויז א געפיל. די פירמע Qodo האט ארויסגעגעבן א גרויסן באריכט פון דער אינדוסטריע, "the state of AI code quality". דער נומער וואס שטעכט ארויס: 76% פון פראגראמירער האבן געזאגט אז זיי גלייבן נישט קאוד וואס AI האט געשריבן אויב קיין מענטש האט עס נישט איבערגעקוקט. דער טעם: וייב-קאוד פראיעקטן זענען אפט גוט אויס אויפן ערשטן בליק, אבער זיי צעברעכן זיך אונטער דרוק, און עס נעמט מער צייט זיי צו פאררעכטן ווי צו בויען פון אנהייב.

דער עיקר פראבלעם מיט וייב-קאודינג איז אז עס איז געבויט אויף אינטואיציע און נישט אויף כוונה. מען האפט אז דער AI וועט עס טרעפן, מען נעמט אן אז די סטרוקטור איז אין ארדנונג, און מען היפט איבער דאס שווערע טראכטן. דער קורס ברענגט א באקאנטן ווערטל: אינטואיציע וואקסט נישט, סטרוקטור יא. אויסגעשטעלט אויף א פשוטן אופן: וייב-קאודינג איז AI פלוס טרעפעניש, און קאנטעקסט-אינזשענירינג איז

AI פלוס פלאנירונג, קלארקייט און סטרוקטור. אויב וייב-קאודינג איז ארבעטן אויפן אויג און קאנטעקסט-אינזשעניריינג איז קלוג בויען, וואו פאסט אריין פראמפט-אינזשעניריינג?

פראמפט-אינזשעניריינג קעגן קאנטעקסט-אינזשעניריינג (6:22:41)

דער עיקר חילוק: פראמפט-אינזשעניריינג איז ווי צו בעטן פון איינעם א טובה אין איין זאץ. קאנטעקסט-אינזשעניריינג איז ווי צו געבן אים א טעקע מיט אלעס וואס ער דארף כדי צו טון די טובה גוט, און נישט בלויז איינמאל נאר אסאך מאל.

דער משל פון א סענדוויטש: ווען מען וויל אז איינער זאל אייך מאכן א סענדוויטש, זאגט מען בייַ פראמפט-אינזשעניריינג נאר "מאך מיר א סענדוויטש", אן שום פרטים. ער ווייסט נישט צי איר זענט אלערגיש צו ערד-ניסלעך, צי איר האט ליב mayo, אדער צי איר זענט א וועגאן. מען וועט מסתמא באקומען א סענדוויטש, אבער אפשר נישט דער וואס מען האט געוואלט. בייַ קאנטעקסט-אינזשעניריינג גיט מען אים א צעטל: "איך בין א וועגאן, קיין ציבעלעס, ראסטל דאס ברויט, נוץ דעם סאוס פון אויבערשטן פאליצע, אט אזוי האב איך ליב מען זאל עס שניידן." איצט מאכט ער נישט סתם א סענדוויטש נאר אייער סענדוויטש, פונקט ווי איר ווילט.

א פארגלייך אין AI-שפראך, פונקט נאך פונקט:

- דער פאקוס: בייַ פראמפט-אינזשעניריינג אויף דער פראגע אליין. בייַ קאנטעקסט-אינזשעניריינג אויף אלעס ארום דער פראגע: דאטא, כללים, געצייג און זכרון.
- די געוועלטיקע גרייס: א פראמפט איז איין ביז דריי שורות. קאנטעקסט באשטייט פון עטליכע פיילס, איינשטעלונגען, ביישפילן און ריכטיגע אנווייזונגען.
- דער ציל: א פראמפט צילט צו איין אנשטענדיגן ענטפער. קאנטעקסט צילט צו א פארלעסליכער סיסטעם וואס ארבעט איבער עטליכע טריט אדער ארבעטן.
- באנוץ: פראמפט-אינזשעניריינג פאסט פאר געוועלטיקן באנוץ, ווי ChatGPT אדער פראגע-און-ענטפער. קאנטעקסט-אינזשעניריינג פאסט פאר עכטע אפליקאציעס, אויטאמאציע און AI-אגענטן.
- ביישפיל: א פראמפט וואלט געזאגט "שרייב א ריינע Python אפליקאציע". מיט קאנטעקסט לייגט מען צו א סיסטעם-כלל, מען זאגט צו ניצן TypeScript, מען לייגט בייַ API-דאקומענטאציע, מען גיט ביישפיל-קאוד, און מען באשטימט א JSON-פארמאט פאר דעם רעזולטאט.
- פארלעסליכקייט: א פראמפט טרעפט אמאל און אמאל נישט. קאנטעקסט איז מער סטרוקטורירט, מען קען עס פאראויסזען, און עס קען וואקסן.

פראמפט-אינזשעניריינג איז גוט פאר אן איינמאליגע פראגע און פאר שנעל פרובירן. אבער ווען מען בויט א AI-טשעטבאט אדער אן אייגענעם AI-אגענט, דארף מען קאנטעקסט-אינזשעניריינג, ווייל דער AI דארף מער ווי בלויז א פראגע. ער דארף די גאנצע מעשה.

די אינגרעדיענטן פון גוטן קאנטעקסט (6:25:16)

דער AI איז נישט קיין מענטש. ער פארשטייט נישט די זאכן ווי מענטשן, ער ווייסט נאר דאס וואס מען גיט אים יעצט. דערפאר, כדי ער זאל טון א גוטע ארבעט, מוז מען אים געבן די ריכטיגע אינפארמאציע אויפן ריכטיגן אופן. די זאמלונג אינפארמאציע הייסט קאנטעקסט. פונקט ווי בייַ א רעצעפט פאדערט גוטער קאנטעקסט געווייסע אינגרעדיענטן, און אויב מען פארגעסט אפילו איינעם קען דער AI עס פארפוישן:

• סיסטעם-אנווייזונגען: דער גרונט-כללים-ביכל פאר דעם AI, למשל "שרייב תמיד ריינעם קאוד", "נוץ בריטישע ענגליש", "הייב אן יעדן ענטפער מיט 'hi there'". דאס זענען אלגעמיינע כללים וואס דער AI דארף תמיד האלטן, נישט קיין חילוק וועלכע ארבעט מען גיט אים.

• דער אינפוט פונעם באניצער: דער אייגנטליכער פראמפט, די פראגע אדער באפעל, למשל "פאסט צוזאם דעם נייעס-ארטיקל" אדער "בוי א וועטער-אפליקאציע אין Python". פשוט און גלייך, און דאס הייבט אן די ארבעט.

• קורצער זכרון, היינו די שמועס-היסטאריע: וואס מען האט געשמועסט ביז יעצט. למשל, מען בעט "קענסט שרייבן א באריכט?" און דערנאך "מאך עס קירצער". דער AI דארף געדענקען די ערשטע בקשה כדי צו פארשטיין די צווייטע.

• לאנגער זכרון: זכרון פון פריערדיגע סעסיעס אדער אפגעהיטענע באפארצוגונגען. למשל, מען וויל תמיד אז דער AI זאל נישט ניצן געוויסע ווערטער, אדער מען האט אים שוין געזאגט דעם נאמען און די שטעלע. אויב ער געדענקט עס, איז דאס לאנגער זכרון. נישט אלע AI-געצייג האבן דאס נאך, אבער די בעסטע, ווי פארגעשריטענע אגענטן, האבן עס יא.

• וויסן-באזעס: דרויסנדיגע מקורות אינפארמאציע וואס דער AI קען ניצן, ווי דאקומענטן, וועבזייטלעך אדער API'ס. למשל, ביים בויען א געזונט-אפליקאציע פארבינדט מען דעם AI מיט א מעדיצינישער דאטא-באזע אדער א PDF-וועגווייזער. ער זוכט דורך דעם מאטעריאל און גיט קלוגערע ענטפערס.

• דער שטאנד פונעם ארבעטס-פלוס (workflow state): וואו האלט מען יעצט אין א גרעסערן פראצעס. למשל, ביים בויען אן אפליקאציע מיט AI: ערשטער טריט פלאנירן, צווייטער שרייבן קאוד, דריטער טעסטן, פערטער פארריכטן די באגס. אויב דער AI ווייסט אויף וועלכן טריט ער האלט, קען ער זיך בעסער קאנצענטרירן. אויב נישט, פרובירט ער אפשר צו טון אלעס אויף איינמאל, און דאס פאלט אפט דורך.

עס איז דא איין פראבלעם: דער AI קען נישט האלטן אן א שיעור אינפארמאציע. ער האט א גרענעץ וואס הייסט context window. ווי אזוי לייגט מען אריין דאס אלעס אן אים צו צעמישן?

פראבלעמען מיט קאנטעקסט און ווי אזוי זיי צו פארריכטן (6:28:21)

צו געבן דעם AI דעם ריכטיגן קאנטעקסט איז זייער וויכטיג, אבער נישט תמיד גרינג. דער קורס רעכנט אויס עטליכע געווענליכע פראבלעמען:

• צו פיל אינפארמאציע, צו ווייניג פלאץ: דער AI האט א גרענעץ וויפיל ער קען לייענען אויף איינמאל. אויב מען גיט אים צו פיל, פארגעסט ער אדער ווערט צעמישט. די לייזונג: פאסט צוזאם עלטערע אדער ווייניגער וויכטיגע אינפארמאציע, כדי צו מאכן פלאץ פאר וואס איז יא וויכטיג.

• איבערגעפילטקייט מיט אינפארמאציע: אז מען שיט אריין גרויסע שטיקער טעקסט אן סדר, קען דאס דעם AI אריבערוואקסן.

• א פאלשע ארדענונג פון דער אינפארמאציע: אויב די וויכטיגסטע אינפארמאציע איז באהאלטן, קען דער AI זי איבערזען.

• עטליכע מקורות וואס צעמישן: אויב א פראיעקט ניצט פארשידענע דאטא-באזעס אדער געצייג, ווייסט דער AI אפשר נישט וועלכן צו ניצן.

• א צעמישטער זכרון: וועו מען היט אפ צו פיל צופעליגע אינפארמאציע, פארבלאנדזשעט דער AI. די לייזונג: ניצן "זכרון-בלאקן" (memory blocks) צו ארגאניזירן וואס דער AI געדענקט, ווי פאקטן, פריערדיגע שמועסן אדער פעסטע כללים.

די פשוטע תיקונים העלפן אז דער AI זאל געבן בעסערע און גענויערע ענטפערס יעדעס מאל.

א דעמא מיט ChatGPT: געוועלטיגט קעגן אייגענעם GPT (6:29:23)

דערנאך ווערט געוויזן א דעמא אין ChatGPT. ערשט גיט מען א פראמפט: "שאף א פראיעקט-פלאן פאר ארויסלאזן א נייעם וועבזייטל". ChatGPT ענטפערט מיט א פלאן: א בליק אויפן פראיעקט, דער ציל, א צייטפלאן, די מאנשאפט און די פאזעס. די פאזעס ווערן אבער נאר אויסגערעכנט, אן אויסצוטייטשן אין דער טיף און אין פרטים.

נאכדעם ניצט מען א custom GPT כדי צוצולייגן מער קאנטעקסט:

• מען קלאפט אויף דער אפציע צו מאכן אן אייגענעם GPT.

• מען גיט אים א נאמען, למשל "project".

• אין די אנווייזונגען שרייבט מען: "דו ביסט א פראיעקט-מענעדזשער מיט עקספערטיז אין ארויסלאזן וועבזייטלעך. שאף א דעטאלירטן פראיעקט-פלאן מיט טערמינען פאר יעדער פאזע, און נעם אין באטראכט דאס פאלגנדע." מען קען דארט צולייגן נאך אינפארמאציע וועו עס איז נויטיג.

• מען קלייבט אויס א פארגעשריטענע מעגליכקייט וואס איז דארט פאראן, און מען קלאפט "save".

• מען עפנט א נייעם שמועס און שטעלט די זעלבע פראגע: "שאף פאר מיר א פראיעקט-פלאן פאר ארויסלאזן א נייעם וועבזייטל".

דאס מאל ענטפערט ער מיט דעם ציל פונעם פראיעקט, און פאר יעדער פאזע (למשל פאזע איינס) מיט דעם ציל און די ארבעטן. ער גיט אויך ווי פיל טעג יעדע ארבעט וועט נעמען, און די milestones. די פאזעס זענען צעטיילט אין אסאך טיילן, און מען זעט קלאר די ארבעט, די צאל טעג און וועלכע milestone מען דערגרייכט.

אין פארגלייך: דער געוועלטיגער ChatGPT האט געגעבן אן אלגעמיינעם ענטפער, מיט פאזעס און ארבעטן אבער אן ריכטיגע טערמינען און צייטן. דער GPT מיט קאנטעקסט האט געגעבן דעם ציל, די ארבעטן פאר יעדער פאזע, און אויך די טעג. אט אזא סארט ענטפער קען מען ערווארטן וועו מען ניצט קאנטעקסט.

א קורצער איבערגאנג: AI, מאשין-לערנען און דיפ-לערנען (6:31:55)

צום סוף פונעם אפשניט הייבט זיך אן אנדער טייל, וואס ערקלערט די שיכטן פון AI. אינערהאלב AI געפינט זיך מאשין-לערנען. מאשין-לערנען לאזט סיסטעמען דערקענען מוסטערן פון דאטא, אנשטאט מען זאל מוזן אויסדריקליך פראגראמירן יעדן באשלוס. אינערהאלב מאשין-לערנען האט דיפ-לערנען אריינגעברענגט נויראל-נעצן וואס קענען ארבעטן מיט פיל מער קאמפליצירטע אינפארמאציע, ווי בילדער, רעדן און טעקסט. גענעראטיווער AI ברייטערט די מעגליכקייטן נאך מער אויס, און דער אפשניט ענדיגט זיך אינמיטן דעם געדאנק.

פון פרעדיקציע ביז פראמפט-אינזשעניערינג (6:32:25)

פון קלאסיפיצירן צו שאפן: וואס איז דזשענערעטיוו AI (6:32:25)

דער קורס ערקלערט אז ביים טראדיציאנעלן מאשין-לערנען פרעגט מען: צו וועלכער קאטעגאריע געהערט דאס, אדער וועלכן ווערט דארף מען פאראויסזאגן? ביי דזשענערעטיוו AI פרעגט מען אן אנדערע שאלה: שאף עפעס נייעס. א דזשענערעטיוו AI סיסטעם קען ארויסגעבן טעקסט, קאוד, בילדער, אודיא, דיזיין, קיצורים, ערקלערונגען און נאך פילע סארטן אינהאלט.

עס ווערט געגעבן א ביישפיל: טראדיציאנעל מאשין-לערנען קען באשטימען צי אן אימעיל איז ספאם אדער נישט, אבער דזשענערעטיוו AI קען אליין שרייבן דעם אימעיל. אזוי אויך קען א קאמפיוטער-וויזשן מאדעל אידענטיפיצירן די זאכן וואס געפינען זיך אין א בילד, אבער א מולטימאדאלער דזשענערעטיוו AI סיסטעם קען אנקוקן דאס בילד און ערקלערן וואס עס גייט פאר אין אים. דער איבערגאנג פון בעיקר פאראויסזאגן און קלאסיפיצירן צו שאפן און אינטעראקטירן איז איינע פון די סיבות פארוואס דזשענערעטיוו AI איז געווארן אזוי איינפלוסרייך.

די היסטאריע: פון נויראל-נעצן ביז טראנספארמערס (6:32:55)

עס ווערט ערקלערט אז דזשענערעטיוו AI איז נישט באוויזן געווארן איבערנאכט. דער וועג האט זיך אנגעהויבן מיט פריע געדאנקען צי א מאשין קען ארויסווייזן אינטעליגענט פירונג. דערנאך האט מאשין-לערנען דערמעגליכט פאר קאמפיוטערס צו לערנען מוסטערן פון דאטא. נויראל-נעצן און דיפ-לערנען האבן דערמעגליכט צו פראצעסירן אלץ מער קאמפליצירטע אינפארמאציע. ארכיטעקטורן ווי רעקורענטע נויראל-נעצן (RNN) האבן פארבעסערט דאס פראצעסירן פון אינפארמאציע וואס קומט אין א סדר, למשל טעקסט. דערנאך זענען געקומען LSTM נעצן (long short-term memory), וועלכע האבן געהאלפן די סיסטעמען האלטן אינפארמאציע איבער לענגערע סעקווענצן.

אבער שפראך איז נאך אלץ געווען א גרויסע שוועריקייט. דער קורס ברענגט א ביישפיל: "The developer updated the application because it had security issue". כדי צו פארשטיין דעם זאץ ריכטיג, דארף א מאדעל דערקענען די פארבינדונגען צווישן ווערטער אפילו ווען זיי שטייען נישט איינס לעבן דאס אנדערע. דא איז דער אטענשאן-מעכאניזם געווארן זייער וויכטיג: ער לאזט א מאדעל באשליסן וועלכע טיילן פונעם אינפוט פארדינען מער אויפמערקזאמקייט ביים פראצעסירן אנדערע טיילן. דער געדאנק איז שפעטער געווארן דער צענטער פון דער טראנספארמער-ארכיטעקטור. טראנספארמערס האבן פון יסוד געטוישט natural language processing און זענען געווארן די ארכיטעקטור הינטער פילע מאדערנע גרויסע שפראך-מאדעלן. אזוי איז מען אנגעקומען צו סיסטעמען וואס קענען שאפן נאטירלעכע ענטפערס וואס פאסן צום קאנטעקסט, אויף א ריזיגן מאסשטאב.

וואס מיינט LLM, און וויאזוי ארבעט עס מיט טאקענס (6:34:26)

דער קורס צעטיילט דעם טערמין "large language model" (LLM) אויף דריי ווערטער:

- Large – באציט זיך צום ריזיגן מאסשטאב פון די סיסטעמען: זייער טריינירונג-דאטא, די קאמפיוטער-באדערפענישן, און די צאל פאראמעטערס פונעם מאדעל.
- Language – זייער פעאיקייט צו פראצעסירן און שאפן מענטשלעכע שפראך.

- Model – א מאטעמאטישע סיסטעם וואס האט זיך אויסגעלערנט מוסטערן און פארבינדונגען פון דער טרייניר‌ונג-דאטא.

אבער אן LLM פראצעסירט נישט שפראך פונקט ווי א מענטש זעט עס. למשל, אז מען שרייבט אריין "artificial intelligence is changing business", ווערט דער טעקסט, איידער דער מאדעל פראצעסירט אים, צעטיילט אויף קלענערע איינהייטן וואס הייסן טאקענס. א טאקען קען זיין א גאנץ ווארט, א טייל פון א ווארט, א פונקטואציע-צייכן אדער אן אנדערע טעקסט-איינהייט, לויט וועלכער tokenizer מען ניצט. די טאקענס ווערן דערנאך אויסגעדריקט אין צאלן, כדי דער נויראל-נעץ זאל זיי קענען פראצעסירן.

ווען אן LLM שאפט אן ענטפער, שאצט עס בעצם אפ וועלכער טאקען איז דער פאסיגסטער צו קומען ווייטער, לויטן קאנטעקסט וואס עס האט שוין באקומען. עס שאפט איין טאקען, ניצט דאן דעם טאקען צוזאמען מיטן פריערדיגן קאנטעקסט כדי פאראויסצוזאגן דעם קומענדיגן, און אזוי ווייטער, ביז דער ענטפער איז פארטיג. דאס מיינט אז אן LLM איז פון יסוד א פראבאביליסטישע סיסטעם: עס זוכט נישט ארויס איין ענטפער וואס ליגט אייביג אפגעהיט פאר יעדע שאלה, נאר עס שאפט א רעזולטאט לויט מוסטערן וואס עס האט זיך אויסגעלערנט ביים טריינירן, און לויט דער אינפארמאציע וואס געפינט זיך אינעם איצטיגן קאנטעקסט. דאס ערקלערט אויך פארוואס א געטוישטער פראמפט קען ברענגען אן אנדער ענטפער.

אטענשאן: ווי מען לערנט פון א לערנבוך (6:36:28)

דער קורס קומט צוריק צו דער שאלה פארוואס טראנספארמער-מאדעלן זענען געווארן אזוי וויכטיג. איינע פון די הויפט-סיבות איז אטענשאן. א ביישפיל-זאץ: "Ria gave Priya the laptop because she needed it for the presentation". דער מאדעל מוז פארשטיין עטליכע פארבינדונגען: אויף וועמען באציט זיך "she"? אויף וואס באציט זיך "it"? וועלכע ווערטער זענען די וויכטיגסטע כדי צו פארשטיין דעם זאץ? דער אטענשאן-מעכאניזם העלפט דעם מאדעל אפשאצן די פארבינדונגען צווישן די פארשידענע עלעמענטן אין דער סעקווענץ. אנשטאט באהאנדלען יעדעס ווארט ווי גלייך וויכטיג, קען דער מאדעל געבן פארשידענע מאס וויכטיגקייט לויטן קאנטעקסט.

עס ווערט געגעבן א משל: ווען מען לערנט פון א לערנבוך, באהאנדלט מען נישט יעדן זאץ ווי גלייך וויכטיג. מען שטרייכט אונטער דעפיניציעס, מען צייכנט אן וויכטיגע ארגומענטן, און מען פארבינדט איין באגריף מיט אן אנדערן וואס איז געשטאנען עטליכע פאראגראפן פריער. אטענשאן גיט נויראל-נעצן א מעכאניזם וואס העלפט זיי מאכן ענליכע קאנטעקסט-פארבינדונגען אויף א מאטעמאטישן אופן. מאדערנע טראנספארמער-מאדעלן ניצן דעם מעכאניזם אויף א בריטן פארנעם, און דאס איז איינע פון די סיבות פארוואס זיי קענען זיך באגיין מיט שפראך אויף אזא גרויסן מאס.

די יסודות פון פראמפט-אינזשעניערינג (6:37:30)

ווען מען פארשטייט וויאזוי אן LLM פראצעסירט אינפארמאציע, ווערט פראמפט-אינזשעניערינג מער פארשטענדליך. א פראמפט איז נישט פשוט א שאלה; עס איז די אינפארמאציע און אנווייזונגען וואס מען גיט דעם מאדעל, כדי ער זאל פארשטיין וואס מען וויל.

דער קורס פארגלייכט צוויי פראמפטס. דער ערשטער: "write about AI". דער צווייטער: "explain generative AI to a beginner in approximately 200 words. Use simple English, provide one business example, and end with three key takeaways". ביידע רעדן וועגן דעם זעלבן נושא, אבער דער צווייטער גיט דעם מאדעל אסאך מער נוצליכע הדרכה. דער ערשטער איז א שוואכער פראמפט, דער צווייטער א שטארקער.

א שטארקער פראמפט גיט געווענליך איבער פיר וויכטיגע זאכן:

- קאנטעקסט – מיט וואספארא מצב האט מען צו טון?
- די אויפגאבע – וואס זאל דער מאדעל טון?
- באגרעניצונגען – וועלכע כללים זאל ער האלטן?
- פארמאט פונעם רעזולטאט – וויאזוי זאל דער ענדגילטיגער ענטפער אויסזען?

מען קען אויך ספעציפיצירן דעם עולם, דעם טאן, די לענג, ביישפילן, טערמינאלאגיע און סטרוקטור. א ביישפיל: "You're explaining machine learning to a first year college student. Explain supervised and unsupervised learning using simple language. Give one example of each and present the final comparison in a table". איצט האט דער מאדעל אסאך מער אינפארמאציע וועגן דעם געוונטשענעם רעזולטאט. דאס איז דער יסוד-פרינציפ פון פראמפט-אינזשעניערינג: וואס קלארער די אנווייזונג, אלץ גרינגער איז פארן מאדעל צו שאפן א נוצליכן ענטפער.

Zero-shot, One-shot און Few-shot (6:39:33)

דער קורס גייט איבער צו פארשידענע מעטאדן פון פראמפטן:

- Zero-shot – מען גיט דעם מאדעל אן אנווייזונג אן קיין שום ביישפיל. למשל: "write a professional LinkedIn post announcing the launch of a new analytical product". דא איז דא אן אנווייזונג, אבער נישטא קיין ביישפיל.
- One-shot – מען גיט איין ביישפיל וואס ווייזט דעם מאדעל וואספארא ענטפער מען ערווארטעט. למשל מען שרייבט "here is an example of the style I want", מען גיט א ביישפיל-פאסט, און ערשט דערנאך בעט מען דעם מאדעל צו שאפן נאך איינס.
- Few-shot – אנשטאט איין ביישפיל גיט מען עטליכע, וואס שטעלט אוועק א קלארערן מוסטער.

עס ווערט באטאנט אז די ביישפילן טריינירן נישט איבער דעם מאדעל אליין; מען גיט אים בלויז נאך אינפארמאציע אינעם איצטיגן קאנטעקסט. דאס רופט מען in-context learning. אויב עס איז וויכטיג צו האלטן א קאנסיסטענטן טאן, סטרוקטור, קלאסיפיקאציע אדער פארמאט, קענען ביישפילן שטארק קלאר מאכן וואס מען ערווארטעט. דער קורס רעקאמענדירט שטענדיג צו ניצן few-shot, ווייל וואס בעסער דער פראמפט, אלץ בעסער דער רעזולטאט.

צעטיילן אין שטאפלען, self-consistency און א בוים פון וועגן (6:40:34)

טייל פראבלעמען קען מען נישט גוט באהאנדלען מיט איין ברייטער אנווייזונג. ביי קאמפליצירטע אויפגאבעס איז כדאי צו צעטיילן די ארבעט אין שטאפלען. למשל ביים בויען א פראדוקט-סטראטעגיע: אנשטאט צו בעטן "create a complete strategy for launching a product", צעטיילט מען:

- ערשט, אידענטיפיצירן די ציל-באניצער.
- דערנאך, לויט די באניצער, אידענטיפיצירן זייערע הויפט-פראבלעמען.
- דערנאך, לויט די פראבלעמען, פארשלאגן דריי positioning-צוגאנגען.
- צולעצט, פארגלייכן די צוגאנגען און רעקאמענדירן דעם בעסטן.

דער רעזולטאט פון יעדן שטאפל גיט קאנטעקסט פארן קומענדיגן. דאס מאכט דעם פראצעס גרינגער צו קאנטראלירן און פארבעסערן: אויב עפעס גייט שלעכט, קען מען פונקטליך אידענטיפיצירן וועלכער שטאפל דארף פארבעסערונג, אנשטאט איבערצושרייבן די גאנצע בקשה. דאס איז די מעלה פון קייט-באזירטן ריזאנינג.

א צווייטער נוצליכער געדאנק איז self-consistency, פאר אויפגאבעס וואס פאדערן שטארק טראכטן: אנשטאט זיך גלייך פארלאזן אויף איין געשאפענע לייזונג, פרובירט מען אויס עטליכע קאנדידאט-צוגאנגען און מען פארגלייכט זיי איידער מען באשליסט דעם ענדגילטיגן רעזולטאט. דער פרינציפ איז פשוט: אויב א פראבלעם קען מען אנגיין אויף עטליכע אופנים, קען א פארגלייך ברענגען א מער פארלעסליכע מסקנא.

דערנאך קומט בוים-באזירטע אויספארשונג: אנשטאט גיין נאר אויף איין וועג, פארשט מען אויס דעם פראבלעם דורך עטליכע מעגליכע צווייגן. למשל, ווען מען וויל פארמינערן קונה-אפגאנג (churn): איין וועג קען אויספארשן פראדוקט-פראבלעמען, א צווייטער – פרייזן, א דריטער – קונה-סופארט, און א פערטער – onboarding. יעדער צווייג איז א מעגליכע ריכטונג, און די נוצליכסטע ריכטונגען קען מען ווייטער אפשאצן. אזא סטרוקטורירטע אויספארשונג איז ווערטפול פאר פלאנירן, פראבלעם-לייזן, סטראטעגיע און אנדערע קאמפליצירטע אויפגאבעס וואו עס איז נישטא קיין קלארער וועג גלייך פון אנהייב.

וויאזוי פארבעסערט מען א פראמפט (6:43:08)

וואס טוט מען ווען דער AI ענטפער איז נישט גוט גענוג? דער קורס זאגט: נישט גלייך אננעמען או דער מאדעל קען נישט דורכפירן די אויפגאבע. ערשט קוקט מען איבער דעם פראמפט, און מען פרעגט זיך:

- האב איך קלאר ערקלערט דעם ציל?
- האב איך געגעבן גענוג קאנטעקסט?
- האב איך ספעציפיצירט דעם געצילטן עולם?
- האב איך קלאר ערקלערט דעם פארמאט פונעם רעזולטאט?
- וואלט א ביישפיל געהאלפן?
- איז די אויפגאבע צו גרויס, און דארף מען זי צעטיילן אין קלענערע שטאפלען?

א ביישפיל: אנשטאט שרייבן "improve my résumé", שרייבט מען: "review the following résumé for a data analyst role. Identify the five weakest bullet points. Rewrite each using action verbs, measurable impact where supported and concise professional language. Do not invent achievements". דאס גיט דעם מאדעל אסאך קלארערע גרעניצן. פראמפט-אינזשעניערינג, ענדיגט דער טייל, איז ווייניגער געפינען א געהיימע זאמלונג פון "צויבער-ווערטער", און מער איבערגעבן די פאדערונגען קלאר און סיסטעמאטיש – און דאס איז די פעאיקייט וואס מאכט דזשענערעטיוו AI נוצליכער. פון דא און ווייטער גייט דער קורס איבער צו א טיפערער לעקציע ביי די טריינירונג-מומחים.

דזשענערעטיוו AI אין דער הייראַרכיע פון AI (6:44:44)

דער נייער טייל הייבט אן מיט א שאלה צום עולם: ווי וואלט מען ערקלערט פאר א פשוט'ן מענטש וואס איז דזשענערעטיוו AI, וואס איז ChatGPT, און וואס זענען גרויסע שפראך-מאדעלן?

דערנאך ווערט געצייכנט א בילד פון שיכטן, איינס אינעם אנדערן: אינעווייניג איז דזשענערעטיוו AI, ארום דעם – דיפ-לערנען, ארום דעם – מאשין-לערנען, און דרויסן – AI. אינערהאלב פון GenAI געפינען זיך די

גרויסע שפראך-מאדעלן. דערפאר קען מען זאגן אז דזשענערעטיוו AI איז א סאבסעט פון AI, ווייל AI איז דאס גרעסערע בילד און נעמט אריין מאשין-לערנען, דיפ-לערנען און דזשענערעטיוו AI.

דיפ-לערנען, אין פשוטע ווערטער, ווערט גענוצט צו באהאנדלען קאמפליצירטע אויפגאבעס און קאמפליצירטע דאטא, ווי בילדער און טעקסט. דאס טוט מען מיט CNN – וואס ארבעטן מיט filters, kernels און convolutions – און מיט RNN, וואס מען קען ניצן צו אנאליזירן time series דאטא און טעקסט-דאטא. אויבן דערויף זענען דא טראנספארמער-מאדעלן, וואס ליגן צווישן דיפ-לערנען און דזשענערעטיוו AI. א טראנספארמער איז א דיפ-לערנען ארכיטעקטור וואס נעמט אריין אן encoder-decoder פראצעס. דער קורס זאגט אז מען גייט דא נישט אריין טיף אין טראנספארמערס; דער פאקוס ווערט זיין אויף דזשענערעטיוו AI און אירע אנווענדונגען.

מאשין-לערנען, ווערט דערמאנט, איז סטאטיסטישע אנאליז – די מאדעלן וואס מען האט שוין פריער געמאכט מיט pandas, NumPy און Seaborn.

פון קלאסישן AI ביז געזעראטיוון AI (6:48:19)

וואס מאשין-לערנען און דיפ-לערנען האבן געקענט טון (6:48:19)

דער קורס הייבט אן מיט א חזרה אויף דעם וואס מען האט שוין דורכגענומען. ביים קלאסישן מאשין-לערנען האט מען געהאט געוויסע לייברעריס וואס מען האט געוואלט צו פראצעסירן און צוגרייטן די דאטא. נאכדעם האט מען געקענט אנווענדן איינעם פון די מאשין-לערנען אלגאריטמען, למשל:

- logistic regression
- linear regression
- XGBoost
- bagging און boosting
- random forest

עס ווערט ערקלערט אז די אלגאריטמען ווערן גענוצט פאר צוויי סארטן אויפגאבעס: classification, ד.ה. איינטיילן אין קאטעגאריעס, און regression, ד.ה. פאראויסזאגן א נומערישן ווערט. דער הויפט-ציל איז געווען צו קענען פאראויסזאגן א רעזולטאט פאר א נייעם ביישפיל ("sample").

ביים דיפ-לערנען האט מען אויך געהאט classification און regression. דערצו האט דיפ-לערנען געקענט סאלווען נאך אויפגאבעס, ווי למשל text summarization, ד.ה. פארקירצן א לאנגן טעקסט, און sentiment analysis, ד.ה. פעסטשטעלן צי א טעקסט איז פאזיטיוו אדער נעגאטיוו.

ווי אזוי געזעראטיוו AI האט געטוישט דאס בילד (6:49:00)

לויט ווי דער קורס ערקלערט, האט די אריינקומעניש פון געזעראטיוו AI (GenAI) געטוישט דאס גאנצע בילד. ביי די פריערדיגע מעטאדן האט מען געדארפט א באזונדערן מאדעל פאר יעדע אויפגאבע. איצט נעמט מען א טעקסט, מען גיט אים אריין אין געזעראטיוון AI מאדעל, און דער זעלבער מאדעל קען סאלווען א סך פארשידענע אויפגאבעס:

- classification

- summarization
- sentiment analysis
- קאוד שאפן (code generation)
- פאררייכטן קאוד (code fixing)

אלע די אויפגאבעס קען מען טון מיט א גענעראטיוון AI מאדעל.

וואס מיינט "גענעראטיוו" – דער נעקסטער טאקען (6:50:20)

עס ווערט ערקלערט אז אזא מאדעל איז אין גרונט א פאראויסזאג-מאדעל (predictive model). זיין ציל איז צו געבן דעם נעקסטן טאקען, וואס מען רופט "next token generation". פון דעם קומט דער נאמען "גענעראטיוו": דער מאדעל שאפט (generate) א נייעם טעקסט, אדער ריכטיגער, דעם נעקסטן טאקען.

דער קורס ברענגט א ביישפיל. מען גיט אריין א טעקסט וואס איז נישט פארענדיגט: "I want to go to _", ד.ה. "איך וויל גיין צו _". דער גענעראטיוער מאדעל גיט א לאנגע רשימה פון פארשלאגן ווי צו פארענדיגן דעם זאץ:

- to a school – אין א שולע
- to a college – אין א קאלעדזש
- to an office – אין אן אפיס
- to a market – אין א מארק

יעדער פון די ווערטער ווערט באטראכט ווי א טאקען, און יעדער טאקען האט א געוויסע פראבאבילקייט (probability). אינעם ביישפיל ווערן געבראכט נומערן ווי 96, 0.5, 40 און 30. פון דער גאנצער רשימה נעמט דער מאדעל דעם טאקען וואס האט די העכסטע פראבאבילקייט, אינעם ביישפיל דעם מיט 96.

דערפאר הייסט עס א פאראויסזאג-מאדעל וואס זוכט דעם נעקסטן טעקסט אדער דאס נעקסטע וואס קומט. דאס איז אויך פארוואס מען רופט עס א probabilistic model, א מאדעל וואס ארבעט לויט פראבאבילקייטן.

דער אונטערשייד צווישן טעקסט און טאקען (6:51:58)

דער קורס ווארנט מען זאל נישט צומישן טעקסט מיט טאקען. טעקסט איז דאס ווארט אליין, למשל "I". א טאקען איז א נומערישע פארשטעלונג פון יענעם טעקסט. אינעם ביישפיל קען מען פארשטעלן דאס ווארט "I" מיטן נומער 56. דער טעם איז אז א קאמפיוטער פארשטייט די שפראך פון נומערן, נישט די שפראך פון ווערטער.

אינעווייניג, ווען מען רעדט פון קאוד, ווערט דער טעקסט קודם איבערגעביטן אויף טאקענס, און די טאקענס ווערן אריינגעגעבן אינעם גענעראטיוון AI מאדעל. אויפן סמך פון די טאקענס רעכנט דער מאדעל אויס פראבאבילקייטן. דער רעזולטאט וואס ער גיט ארויס איז דער טאקען וואס האט די העכסטע ווערט פון דעם גאנצן "קארפוס" פון טאקענס.

ווען מען גיט מער טעקסט, שאפט דער מאדעל ווייטער. וועלכע אויפגאבע עס איז, צי summarization , צי classification, צי code generation, דאס ענגט אפ אין דער אנווענדונג.

דער קורס באמערקט אז דאס איז א זייער ברייטע טעמע, און דא גיט מען בלויז א גרונטלעכן באגריף ווי אזוי א געניראטיוו מאדעל ארבעט. שפעטער וועט מען זען די אנוועדונגען, ווי למשל ChatGPT, וואס איז א פראמפט, ווי אזוי מען שרייבט פארשידענע פראמפטס, און וועלכע רעזולטאטן מען באקומט.

דאס אלגעמיינע בילד איז אזוי: מען גיט אריין אן אינפוט אין דער פארעם פון א פראמפט, און דער געניראטיוו מאדעל קען דורכפירן א סך אויפגאבעס און טעטיקייטן. אין די קומענדיגע לעקציעס וועט מען מער ערקלערן ווי אזוי געניראטיוו AI ארבעט און וואס איז די טעאריע הינטער דעם.

די ציילן פון דער לעקציע: פראמפטס און אנוועדונגען (6:54:30)

עס ווערט דערציילט וואס מען וועט קענען ביזן סוף פון דער לעקציע. ערשטנס וועט מען אידענטיפיצירן אנוועדונגען פון GenAI וואס סאלווען פראבלעמען אין דער פאקטישער וועלט. צווייטנס וועט מען שאפן עפעקטיווע פראמפטס.

א פראמפט, לויט ווי דער קורס ערקלערט, איז דער אופן ווי מען זאגט קלאר פאר דעם GenAI וואס פונקט מען וויל. עס ווערט געבראכט א משל: שרייבן א פראמפט איז ווי מען רעדט צו א קינד פון צוויי יאר. דער פראמפט מוז זיין קריסטאל-קלאר, מיט א קלארן סעט אנווייזונגען, צי מען ארבעט מיט ChatGPT צי מיט אן אנדער GenAI. אין קאפ האט מען א סך זאכן, אבער וועז מען גיט עס פאר א GenAI מאדעל, מוז מען שרייבן פונקטלעך וואס מען וויל.

דער קורס ווייזט אויף פרטים וואס זענען אויך וויכטיג:

- אפילו א שפאציע (spacing) אינעם פראמפט ווערט געראכנט.
- א פונקט ווערט געראכנט.
- ספעציעלע אותיות ווערן געראכנט.
- די סטרוקטור, ד.ה. אין וועלכער פארעם מען גיט די אינפארמאציע: פשוטער טעקסט, JSON אדער XML.

וועלכע פארעם איז בעסער, דאס איז אויך א טייל פון א גוטן אופן צו קאמוניקירן מיטן געניראטיוון מאדעל, און דאס וועט מען דורכשמועסן.

ווייטער וועט מען שאפן פערזענליכע דאקומענטן, למשל resumes, און דורכפירן פארשונג (research).

מען וועט אויך זען אפליקאציעס וואס זענען געבויט אויף ChatGPT, ווי Canva, Designer GPT, Consensus און Universal Primer. דאס זענען כלים וואס GPT שטעלט צו פאר באזונדערע אויפגאבעס:

- Canva – פאר דיזיין.
- Designer GPT – פאר דיזיינען וועבסייטן.
- Consensus און Universal Primer – פאר חינוך און פארשונג.

מען וועט זען די פראקטישע אנוועדונג פון די אלע כלים, מען וועט שרייבן פראמפטס, און מען וועט ווייזן ווי אזוי מען ארבעט מיט די סיסטעמען.

די אנטוויקלונג: Turing טעסט און ELIZA (6:57:10)

דערנאך גייט דער קורס איבער צו דער היסטאריע, ווי אזוי AI האט זיך אנטוויקלט ביז געניראטיוו AI.

אין די 1950ער יארן איז געווען דער Turing טעסט. דער קורס באשרייבט אים אזוי: אין א צימער זיצט א מענטש, און אינדרויסן זיצט נאך א מענטש. אינעווייניג ווערט ארויסגעשיקט א קול, א מאל פון א מענטש און א מאל פון א מאשין, און דער מענטש אינדרויסן דארף אידענטיפיצירן צי דער קול קומט פון א מאשין אדער פון א מענטש.

דער געדאנק איז אז די מאשין פרובירט נאכצומאכן א מענטש. אויב דער מענטש אינדרויסן אידענטיפיצירט ריכטיג אז עס איז א מאשין, איז די מאשין דורכגעפאלן דעם Turing טעסט. אויב ער אידענטיפיצירט נישט ריכטיג, האט די מאשין דורכגעהאלטן דעם טעסט. די מאשין פרובירט צו "פאלשן" א אמת'ן מענטש.

אין 1966 איז געקומען דער chatbot ELIZA. ער האט גענוצט שפראך-פראצעסירונג מיט מוסטערן (patterns), און ער האט נאכגעמאכט מענטשליכע שמועסן דורך pattern matching און pattern substitution. דאס האט געגעבן פארן באניצער א דמיון ווי ער וואלט פארשטאנען, אבער ער האט נישט געהאט קיין קאנטעקסטועלע אינפארמאציע.

די תקופה פון נויראל-נעצן: AlexNet און CNN (6:59:50)

אין די שפעטע 1980ער יארן איז אנגעקומען די תקופה פון נויראל-נעצן. דער באגריף איז געגומען געווארן פון די נוראנען אינעם מוח, און דער ציל איז צו סאלווען קאמפליצירטע פראבלעמען. דער ביישפיל: מען האט א בילד און מען וויל דערקענען צי דאס איז א הונט אדער א קאץ. א מענטש קען דאס דערקענען, און די פראגע איז צי א מאשין קען דאס אויך.

לויט ווי דער קורס דערציילט, איז דער ערשטער גרויסער דורכברוך געווען AlexNet, וואס האט דערקענט נומערן. מען האט גענוצט א CNN, א שיכט פון א נויראל-נעץ, און מען האט באוויזן אז א מאשין קען דערקענען א מוסטער אדער א נומער וועו מען גיט אריין א בילד אינעם נויראל-נעץ. למשל, אויב אויפן בילד שטייט דער נומער 5, דערקענט דער נעץ אז עס איז 5.

דאס איז געווען א supervised אויפגאבע. מען האט פריער אריינגעגעבן א סך בילדער, און מען האט טריינירט דעם נויראל-נעץ מאדעל צו דערקענען אזא בילד.

RNN און LSTM: ארבעטן מיט סעקוענצן (7:01:20)

דערנאך זענען געקומען RNN און LSTM, וואס זענען געבויט געווארן צוליב די חסרונות פון CNN. א CNN האט געקענט פראצעסירן בלויז דאטא מיט א פעסטער לענג. א בילד האט א פעסטע גרייס, למשל 5×5, 3×3 אדער 256×256, און דאס איז געווען א באגרעניצונג.

ביי RNN, אויף דער אנדערער זייט, קען מען האבן טעקסט אדער time series. מען האט "states", מצבים, וואו די אינפארמאציע ווערט איבערגעגעבן פון איין מצב צום נעקסטן. אזוי קען מען דורכפירן classification אדער sequence-to-sequence, ד.ה. שאפן א סעקוענץ פון א סעקוענץ. LSTM מיינט Long Short-Term Memory networks.

GRU, attention און א ביישפיל פון איבערזעצונג (7:02:58)

אין 2014 זענען געקומען GRU און דער attention mechanism. דער attention mechanism איז שפעטער, אין 2017, אריינגעשטעלט געווארן אינעם transformer מאדעל.

עס ווערט ערקלערט אז RNN האט גוט געארבעט. א ביישפיל פון אן אויפגאבע וואס RNN האט געטון: מען וויל איבערזעצן דעם זאץ "I want to go to market" אויף Hindi, "Main market ja raha hoon". דער קורס הייבט אן ערקלערן אז דאס איז א ביישפיל פון אזא סארט אויפגאבע, און דא ענדיגט זיך דער אפשניט.

פון RNN ביז LLM: אטענשאן און טראנספארמערס (7:03:59)

RNN, ענקאדער און דעקאדער ביי איבערזעצונג (7:03:59)

דער קורס ערקלערט ווייטער ווי מען האט אמאל געמאכט איבערזעצונג פון איין שפראך צו א צווייטער מיט RNN. מען זעט א בילד: איין RNN-בלאק באקומט דעם אינפוט, און די "סטעיטס" (דער אינערליכער צושטאנד) וואס ער אריסגיט ווערן איבערגעגעבן צו א צווייטן RNN, וואס פרובירט עס איבערצודרייען.

דער ערשטער טייל הייסט ענקאדער-בלאק (encoder): ער "ענקאדירט" די דאטא. דער רעזולטאט גייט צו א דעקאדער-בלאק (decoder), וואס "דעקאדירט" די אינפארמאציע, דערקענט מוסטערן, טוט א גענויסע פראצעסירונג, און דרייט איבער דעם ענגלישן טעקסט אויף א הינדי טעקסט. ביידע טיילן זענען נויראל-נעצן.

דער ביישפיל אין דער לעקציע איז דער זאץ "I want to go to the market", וואס דארף ווערן איבערגעזעצט אויף הינדי. דאס איז א מאשין-איבערזעצונג, וואו יעדעס טייל פונעם זאץ דארף ווערן איבערגעדרייט אויף א צווייטער שפראך לויט דעם קאנטעקסט.

די שוואכקייט, און די אטענשאן-מעכאניזם (7:05:00)

עס ווערט ערקלערט אז דער ענקאדער-דעקאדער מיט RNN האט געהאט פראבלעמען: די איבערזעצונג איז נישט געווען גענוג גוט, ווייל די מאדעל האט נישט אונטערגעשטראכן וועלכע ווערטער זענען באמת וויכטיג פאר דער איבערזעצונג. צום ביישפיל: איז "want" וויכטיג, אדער "market"? וועלכעס פון זיי דארף באקומען א העכערע וואג?

דעם פראבלעם האט געלייזט די אטענשאן-מעכאניזם (attention mechanism), וואס גיט יעדן וואָרט א געוויסע וואג (weight). דער ביישפיל אין דער לעקציע:

- "I" קען זיין וויכטיג. אויב אין דעם קומענדיגן זאץ שטייט "he", זענען "I" און "he" פארבונדן, און מען ווייסט אז ביידע רעדן וועגן דער זעלבער פערזאן. דעריבער באקומען זיי א העכערע וואג.
- "to" איז נישט אזוי וויכטיג, און באקומט א נידעריגערע וואג ווי "I".
- "market" קען יא זיין וויכטיג. אין פארגלייך מיט "to" באקומט "market" א העכערע וואג.
- "go" און "want" קענען אויך זיין וויכטיג.

אזוי קען מען, מיט דער אטענשאן-מעכאניזם, זיך קאנצענטרירן אויף די וויכטיגסטע ווערטער אין א זאץ, און דאס איבערגעבן צום דעקאדער.

דער משל פון אונטערשטרייכן א דאקומענט (7:07:01)

צו מאכן עס קלאר, גיט דער קורס א משל פון א מענטש וואס לייענט א דאקומענט. ביים לייענען מארקירט מען (highlight) געוויסע זאצן: "דער טייל איז וויכטיג, דאס קען קומען אויפן עקזאם, דעם באגריף דארף איך לייענען און אויסלערנען". אזוי טראכט א מענטש.

דאס זעלבע טוט מען מיט מאשינען: מען גיט זיי א וואג-סיסטעם, וואס זאגט אז געוויסע ווערטער זענען באמת וויכטיג און באקומען א העכערע וואג. וועז מען גיט עס ווייטער צו א צווייטע סיסטעם, ווייסט די סיסטעם אויף וועלכן טעקסט זיך צו קאנצענטרירן, און נישט אויף דעם איבעריגן. דערפאר ווערט די איבערזעצונג בעסער: אנשטאט צו קוקן אויפן גאנצן דאקומענט, קוקט מען נאר אויף די טיילן מיט העכערער וואג.

דער קורס גייט נישט אריין אין דער מאטעמאטיק און די פארמולעס פון אטענשאן. דער עיקר: וועז מען גיט אריין א זאץ, באקומען די ווערטער מיט העכערער וואג, די וויכטיגע ווערטער, מער "אויפמערקזאמקייט" וועז זיי גייען צום דעקאדער. עס ווערט באמערקט אז אויב מען האט עס נאר טיילווייז פארשטאנען, איז אויך גענוג. דער גאנצער פראצעס וואלט אוועקגעפירט פון דעם הויפט-ענין, וואס איז Generative AI און אירע געצייג. מען דארף נאר א ביסל טעאריע צו פארשטיין.

טראנספארמערס און זייערע דריי סארטן (7:09:06)

נאך דער אטענשאן-מעכאניזם זענען געקומען די טראנספארמערס (transformers), געבויט אויף אטענשאן. דער גרויסער דורכברוך איז געווען דער ארטיקל "Attention Is All You Need". פון דארט האט זיך אנטוויקלט דער טראנספארמער מיט ענקאדער-דעקאדער ארכיטעקטור: מען גיט אריין טעקסט אין ענקאדער, און עס קומט ארויס אן איבערזעצונג.

דער קורס שטעלט אוועק דריי סארטן טראנספארמערס:

- נאר ענקאדער (encoder only): מען גיט אריין טעקסט און באקומט א קלאסיפיקאציע (classification). דער ביישפיל איז BERT (Bidirectional Encoder Representations from Transformers, וואס איז ארויסגעגעבן געווארן פון Google.
- ענקאדער און דעקאדער צוזאמען: למשל T5, אויך פון Google. T5 קען נישט נאר איבערזעצן, נאר באהאנדלען אסאך פארשידענע אויפגאבעס.
- נאר דעקאדער (decoder only): מען גיט אריין טעקסט און באקומט טעקסט-געענעראציע. דאס איז GPT, דאס הייסט די LLM.

LLM איז א דעקאדער-מאדעל (7:11:38)

עס ווערט אונטערגעשטראכן אז LLM איז אייגנטליך א טראנספארמער, און פון די דריי קאטעגאריעס געהערט LLM צו "נאר דעקאדער". זי איז געמאכט פאר טעקסט-געענערציע. דאס איז א פראבאביליסטישע מאדעל (פון וואַרשיינלעכקייט): מען גיט אריין אן אינפוט, און זי גיט ארויס דעם קומענדיגן טאקען (token), דאס קומענדיגע ווארט אדער זאץ. דאס איז וואס GPT קען.

דער קורס פאסט צוזאם די גאנצע אנטוויקלונג: פונעם Turing Test צו נויראל-נעצן, דערנאך RNN, LSTM, GRU, דערנאך דער דורכברוך פון אטענשאן, וואס האט געברענגט די טראנספארמערס מיט די דריי סארטן.

אלע באקאנטע מאדעלן, ChatGPT, Gemini, Grok, Claude און נאך, זענען "נאר דעקאדער" מיט קלייניע ענדערונגען, און זענען טריינירט געווארן אויף מיליאנען און מיליאנען דאטא.

די GPT-סעריע און די פארמעסט (7:13:11)

אויף דעם יסוד האט זיך אנטוויקלט די GPT-סעריע: GPT-1, דערנאך 2, 3, 4, GPT-4o און ווייטער, און עס גייט נאך אלץ ווייטער. דער קורס באמערקט אז נישט אלע לעצטע ווערסיעס זענען אפשר אריינגערעכנט, ווייל די

פארמעסט נעמט נישט קיין סוף: יעדע פירמע מאכט אירע אייגענע GPT -ס. דאס איז געווען די אנטוויקלונג פון AI צו Generative AI.

מען ערמאנט נאכאמאל, ווי א סטודענט האט אנגעוויזן, אז Generative AI באשאפט נייע אינהאלט, ווי בילדער, טעקסט, מוזיק און אנדערע שעפערישע ארבעט. LLM ווערט דא גערופן א דעקאדער-מאדעל, א טראנספארמער-מאדעל: מען גיט אריין אינפוט, און זי גיט ארויס אן אויספיר.

פאראמעטערס, גרייס און טריינינג-דאטא (7:14:43)

א LLM ווערט טריינירט אויף א ריזיגע מאס דאטא. ביי א טריינירטער מאדעל רעדט מען פון צוויי זאכן: די פאראמעטערס, דאס הייסט די גרייס פון דער מאדעל, און די טריינינג-דאטא.

- GPT-2 האט געהאט נאענט צו 1.7 ביליאן פאראמעטערס (אזוי ווי עס ווערט געזאגט אין דער לעקציע).
- GPT-3 האט געהאט 175 ביליאן פאראמעטערס.

פאראמעטערס זענען פשוט די וואגן (weights) פון א מאדעל: די טריינירטע וואגן פון א מאשין-לערנען מאדעל הייסן פאראמעטערס. וואס מער וואגן, אלס גרעסער די מאדעל. די GPT-2 מאדעל איז געווען בערך 16 GB , און GPT-3 איז, לויט דער לעקציע, נאך גרעסער. דער דאטא-סעט אויף וועלכן מען טריינירט GPT-מאדעלן איז ארום 500 GB און נאך מער.

פארוואס מען ניצט API און נישט דעם אייגענעם קאמפיוטער (7:16:18)

דערפאר, ערקלערט דער קורס, קען מען ChatGPT נישט טריינירן אויף אן אייגענעם קאמפיוטער. ביי GPT-2 האט מען ארויסגעגעבן די וואגן (model weights). מען קען זיי אראפנעמען, בערך 16 GB , און לויפן inference) באניצן די מאדעל צו באקומען ענטפערס) אויפן אייגענעם קאמפיוטער. פון GPT-3 און ווייטער האט מען נישט ארויסגעגעבן די וואגן, נאר מען שטעלט זיי צו דורך APIs.

דאס איז וואס מען טוט אין דעם קורס: מען רופט די API פון OpenAI, Gemini אדער Google, מען שרייבט א פראמפט און מען באקומט דעם אויספיר. מען רופט זייערע סערווערס, וואו די מאדעלן ליגן. זיי ליגן נישט ביי אונז, און מען קען זיי נישט אראפנעמען.

GPT-2 קען מען יא לויפן ביי זיך, אבער מען דארף מינדעסטנס 16 GB RAM. וועז מען ארבעט מיט א מאדעל, צי מען טריינירט זי צי מען טוט עפעס אנדערש, ווערט די מאדעל ארויפגעלאדן אין RAM. דעריבער דארף מען צומינדעסט 16 GB RAM צו לאדן אזא GPT מאדעל אויפן לאקאלן קאמפיוטער.

דער אונטערשייד פון פריערדיגן מאשין-לערנען (7:17:51)

דער קורס פארגלייכט: פריער, ביים דיפ-לערנען אדער מאשין-לערנען, האט מען גוט געקענט טריינירן אויף דער אייגענער דאטא, ווייל די דאטא איז געווען קליין, למשל 10,000 ביישפילן, 1 GB אדער 10 GB. דאס איז נישט געווען קיין פראבלעם. ביי Generative AI אבער טריינירט מען אויף ריזיגע מאסן דאטא. ערשטנס קען מען אליין נישט צוזאמנעמען אזויפיל דאטא. צווייטנס דארף מען אויך שווערע הארדווער צו טריינירן. נאכן טריינירן באקומט מען א מאדעל, און די מאדעל האבן די פירמעס שוין צוגעשטעלט דורך APIs. דאס איז דער גאנצער פראצעס, ווי מען ארבעט מיט ChatGPT דורך API.

וואס Generative AI קען טאן, און איר צייל (7:19:00)

די מאדעלן קענען לייזן אסאך אויפגאבעס:

- טעקסט-געגנראציע
- אימעיל-קלאסיפיקאציע
- סענטימענט-אנאליז (sentiment analysis)
- קול-דערקענונג
- מעסעדזשעס און chatbot

דער ציל פון Generative AI, ווי עס ווערט ערקלערט, איז צו באשאפן נייע אינהאלט אויף אן אייגענעם אופן. די געגנראציע איז א פראבאביליסטישע מאדעל, וואס באשאפט נייע אינהאלט דורכן ארויסציען...

געגנראטיוע AI, זארגן, און גרויסע שפראך-מאדעלן (7:19:58)

וויאזוי געגנראטיוע AI ארבעט און וואס עס גיט (7:19:58)

דער קורס ערקלערט אז א געגנראטיוער מאדעל דערקענט מוסטערן (patterns) אין א ריזיגע מאסע דאטא. פון די מוסטערן לערנט ער זיך וויאזוי צו בויען די סטרוקטור פון נייעם אינהאלט, און אזוי שאפט ער נייעם אינהאלט.

א צווייטע מעלה איז אז עס העלפט ביי קאמפליצירטע פראבלעמען. דער מאדעל איז שוין געטרייניירט אויף א גרויסע מאסע דאטא. וועז מען שרייבט אים א פראבלעם, זוכט ער אויס צי אן ענליכער פראבלעם איז שוין אמאל געלייזט געווארן. אויף דעם יסוד גיט ער א פארשיידנקייט פון מעגליכע תוצאות.

א ביישפיל פון טאג-טעגליכן לעבן: מען וויל שרייבן אן אימעיל. מען דערציילט קורץ דעם קאנטעקסט, למשל אז מען וויל בעטן פריי (leave), אדער מען וויל שרייבן צו א פראפעסאר וועגן אייגנעבן א פראיעקט. די AI שאפט א שיין געשריבענעם בריוו וואס פאסט צום קאנטעקסט וואס מען האט אנגעגעבן.

עס ווערט ערקלערט אז דאס האט פארשטארקט די שעפערישקייט, ווייל עס ברענגט ארויס אייגנארטיגע תוצאות. עס שפארט צייט, און עס עפנט נייע אידייען און מעגליכקייטן.

היינט זענען פאראן געצייג פון "no code". דארט דארף מען נישט קענען פראגראמירן. מען זאגט פשוט דעם געצייג אז ער זאל בויען א וועבזייטל אדער שרייבן דעם קאוד, און ער שרייבט עס.

האלוצינאציע, אפהענגיקייט, און א ווארענונג (7:22:00)

ביי דעם אלעם ווערט אונטערגעשטראכן אז מאנכמאל דארף א מענטש זיך איינמישן, ווייל געגנראטיוע AI "האלוצינירט" זייער אסאך.

דערנאך קומט א שאלה פון א באטיילגטער אין קורס: צי וועט דער מענטשליכער שכל און שעפערישקייט נישט ווערן אפהענגיג אין די טשעטבאטס און AI סיסטעמען? דער ענטפער גייט אזוי:

- די וויסנשאפטלעכע פילמען פון די 80ער יארן האבן געשיקט א מעסעדזש, אז מענטש-ענליכע ראבאטן, טשעטבאטס און AI וועלן אונז פארשטויסן און הערשן איבער אונז. מען רעדט אז דאס קען זיך אנהייבן זען אין די קומענדיגע צען יאר.
- אבער מען קען דאס קאנטראלירן, און מען דארף עס ניצן צו זיין פראדוקטיוו.
- קודם זאל מען זען וויאזוי מען קען אויסניצן דעם אייגענעם מוח פאר די וויכטיגסטע אויפגאבעס. ערשט דערנאך זאל מען אריינברענגען די טשעטבאטס.

• אין שעפערישקייט האט עס יא מוסיף געווען. דערפאר דארף היינט יעדער מענטש לערנען די געצייג. יעדע פירמע פארלאנגט אריינצופירן AI לייזונגען, און וועט עס ניצט זיי נישט, וועט בלייבן הינטערשטעליג.

• מען זאל עס ניצן אויף אן אופן וואס הייבט אויף די פראדוקטיוויטעט און שפארט צייט. אין דער געשפארטער צייט קען מען טון אנדערע זאכן.

• דער הויפט-וואַרענונג: מען זאל עס ניצן מיט פארזיכטיגקייט. מען זאל זיך נישט "פארדארבן דעם מוח" און ווערן אינגאנצן אפהענגיג. מען זאל צולייגן אייגענע אידייען וויאזוי צו בלייבן העכער די טעכנאלאגיע.

גרויסע שפראך-מאדעלן: פריער-געטריינירט (7:24:06)

דערנאך גייט דער קורס איבער צו Large Language Models (LLM). דאס זענען פריער-געטריינירטע מאדעלן (pre-trained). מען דארף זיי נישט טריינירן אויף דעם אייגענעם קאמפיוטער. פאקטיש איז עס אוממעגליך פאר א יחיד, צוליב צוויי באגרענעצונגען:

• דאטא: עס פאדערט א ריזיגע מאסע דאטא וואס א יחיד קען נישט צונויפקלייבן.

• האַרדווּעיר: עס פאדערט אסאך GPUs מיט הויכע כח, וואס זענען זייער טייער.

דערפאר זענען עס גרויסע פירמעס וואס בויען די מאדעלן און גיבן זיי ארויס פאר די קונים.

די מאדעלן קענען פארשטיין און שאפן טעקסט ווי א מענטש, אין פארשיידענע אויפגאבעס און געביטן. די טעכנאלאגיע שפרייט זיך שוין אויס צו בילדער, מוזיק און אנדערע זאכן.

"Large", "Language", "Model": דער טייטש פון יעדן וואַרט (7:25:41)

דער קורס צעטיילט דעם טערמין אויף דריי טיילן:

•) Largeגרויס): ווייל די מאדעלן זענען זייער גרויס, למשל 176 ביליאן פאראמעטערס. פאר אזא מאדעל דארף מען ארום 300 GB RAM כדי צו לויפן inference אויף איין סיסטעם.

• א פשוטער חשבון וועט געגעבן: נעם די צאל פאראמעטערס (אין ביליאנען), טאפל עס, און מען באקומט בערך וויפיל GB מען דארף. למשל, א מאדעל פון ארום 1.4 ביליאן פאראמעטערס האט געדארפט ארום 16 ביז 20 GB פאר inference.

• די ציפערן וואס דער קורס דערמאנט באציען זיך אויף ChatGPT 3. ביי ChatGPT 4 איז דער מאדעל אסאך גרעסער, סיי אין גרייס און סיי אין קאמפליצירטקייט.

•) Languageשפראך): די הויפט-פונקציע איז צו פארשטיין און שאפן מענטשליכע שפראך. דאס איז א פראבאביליסטישער מאדעל און א געניראטיוועַר מאדעל. ער איז געבויט אויף א transformer ארכיטעקטור, ספעציפיש א decoder ארכיטעקטור מיט דעם attention mechanism.

•) Modelמאדעל): דאס איז א מאטעמאטישע פארשטעלונג וואס כאפט די מוסטערן און די סטרוקטור פון שפראך-דאטא.

וועגן דער ארכיטעקטור וועלט ערקלערט אז זי קען זיך טוישן פון איין GPT צום אנדערן, און פון פירמע צו פירמע. דער יסוד איז א פשוטער "vanilla" decoder מאדעל. אויף דעם האבן די פירמעס געמאכט ענדערונגען, און אזוי קומען ארויס נייע state-of-the-art מאדעלן.

די באקאנטסטע LLM מאדעלן און וועַר האט זיי ארויסגעגעבן (7:28:45)

די ליסטע פון ברייט-געוונטע מאדעלן:

- ChatGPT (3.5, 4, 4o, mini): פון OpenAI.
- Gemini: פון Google.
- Llama: פון Meta AI.
- Falcon: פון Technology Innovation Institute.
- Claude (v1, v2): פון Anthropic.

API שליסלען, טאקענס, און פריי קעגן באצאלט (7:29:17)

די פירמעס האבן א טשעט-פנים (chat interface), ווי ChatGPT, פאר דירעקטן באנוץ. וועט עס וויל ניצן די מאדעלן אין קאוד, למשל אין Python, אין א front end, אדער אין אן אפליקאציע, באקומט פון זיי API שליסלען (API keys).

דער פראצעס גייט אזוי:

- מען עפנט א חשבון (account) אויפן פארטאל פון דער פירמע.
- מען בעט א API key.
- אין אנהייב איז עס פריי, פאר א געוויסע צייט אדער פאר א געוויסע צאל ערשטע טאקענס.
- וועט די טאקענס ווערן אויסגענוצט, הייבן אן קומען מעלדונגען אז מען דארף באצאלן כדי ווייטער צו ניצן.

די פריי-גרענעצן טוישן זיך לויט יעדן פארטאל, און זיי ווערן כסדר דערהיינטיגט.

פאראן אויך open source געצייג, למשל Ollama, וואס איז פריי. דער קורס זאגט צו אז מען וועט דערויף רעדן אין א שפעטערדיגער לעקציע.

אבער די פרייע געצייג האבן זייערע שווערigקייטן. זיי ווערן אראפגעלאדן אויפן אייגענעם קאמפיוטער, דערפאר זענען זיי אפט פאמעלעך.

וואס קען א LLM טון, און וויאזוי מען רעכנט דעם פרייז (7:31:49)

די הויפט-מעגליכקייטן זענען:

- שאפן אינהאלט.
- צוזאמפאסן טעקסט (summarization).
- אנאליז פון געפילן (sentiment analysis).
- ארויסנעמען דאטא.
- איבערזעצן שפראכן.

א ביישפיל פון צוזאמפאסן: מען האט א קאפיטל פון א בוך און מען וויל עס נישט לייענען אינגאנצן. מען שטופט אריין דעם גאנצן טעקסט, און מען גיט א פראמפט: "גיב מיר א צוזאמפאסונג אין 2,000 ווערטער" (אדער 200). די AI גיט צוריק די צוזאמפאסונג.

אבער דאס קאסט טאקענס. דער אריינגאנג-טעקסט ווערט צעטיילט אין טאקענס, און דער ארויסגאנג-טעקסט אויך. מען רעכנט ביידע צוזאמען.

למשל: 2,000 טאקענס אריינגאנג פלוס 200 טאקענס ארויסגאנג מאכט 2,200 טאקענס. אזויפיל טאקענס זענען אויסגענוצט געווארן.

דאס איז די גאנצע פרייז-סיסטעם פון GPT. ביז א געוויסע צאל טאקענס איז עס פריי, און ווער עס גייט אריבער, באצאלט.

יעדעס ווארט ווערט פארוואנדלט אין טאקענס. אן איינציג ווארט קען זיך צעטיילן אויף עטליכע טאקענס, לויט דעם ווארט. פאראן פארשיידענע מעטאדן ווי אזוי צו פארוואנדלען ווערטער אין טאקענס, און ערשט דערנאך רעכנט מען דעם פרייז.

Gemini, און ווער איז דער בעסטער מאדעל? (7:34:25)

Gemini העלפט ביי קאמפליצירטע פראבלעמען און ביי שאפן קאוד.

אויף דער שאלה ווער איז דער בעסטער (OpenAI, Google, Llama, Falcon אדער Anthropic), ענטפערט דער קורס: ביז מען פרובירט עס נישט אליין, וועט מען קיינמאל נישט וויסן וואס פאסט פאר אייך.

א ביישפיל פון פראקטיק ווערט געבראכט: ביי איין פראיעקט האבן Gemini אדער Claude געגעבן אויסגעצייכנטע רעזולטאטן, און ChatGPT נישט. ביי אן אנדער פראבלעם האט ChatGPT גוט געארבעט, און די אנדערע נישט.

וועלכע AI געצייג צו קלייבן, און א רעזומע מיט ChatGPT (7:35:27)

ווי אזוי קלייבט מען צווישן די פארשידענע מאדעלן (7:35:27)

דער קורס ערקלערט אז עס איז נישטא קיין איין ענטפער וועלכער מאדעל איז דער בעסטער. עס הענגט אפ אין וואס מען דארף. מען קען פרובירן יעדן פון די מאדעלן. יעדע פירמע זאגט אז איר מאדעל ארבעט אויף קאמפליצירטע פראבלעמען און אויף שרייבן קאוד. אנדערע זאגן אז זיי זענען גוט אין איבערזעצן שפראכן אדער אין שאפן אינהאלט. דער באניצער באשטימט אליין וועלכן טייל ער וויל באנוצן.

עס ווערט אנגעווויזן אז OpenAI טוישט שטארק און שנעל אירע מאדעלן, איר טעכנאלאגיע און איר ארכיטעקטור, כדי צו געבן בעסערע רעזולטאטן. Google איז אויך א גרויסע פירמע וואס ברענגט ארויס מאדעלן. יעדן טאג ארבעטן פארשער און געלערנטע אויף לייזונגען פאר קאמפליצירטע פראבלעמען.

דער קורס זאגט צו אז שפעטער, אין צווייטן טייל, וועט מען ווייזן לייוו-דעמאנסטראציעס פון ChatGPT געצייג. ווי מען האט שוין פריער דערמאנט, פאר א דיזיינער איז דא א באזונדער ChatGPT געצייג, און פאר מאכן באריכטן איז דא נאך א געצייג.

נאכדעם ווערן דערמאנט נאך מאדעלן:

- Falcon – גוט פאר אלגעמיינעם שכל-ריזאנירן (common reasoning) און פאר סענטימענט-אנאליז.
- Anthropic און Llama – אין דער לעקציע ווערט געזאגט אז מען באנוצט זיי אין טאג-טעגלעכע ארבעט, צוזאמען מיט ChatGPT. מען נעמט וועלכער מאדעל עס גיט דעם בעסטן רעזולטאט.
- Grok – ווערט באשריבן ווי זייער גוט, ווייל ער גיט א דעטאלירטע ערקלערונג. ער שטיצט אויך אסאך שפראכן.

אלס ביישפיל פאר Grok ווערט דערציילט אז עלטערע מענטשן קענען רעדן צו Grok אויף Hindi. ער פארשטייט Hindi און ענטפערט אויך צוריק אויף Hindi.

די הויפט-עצה איז אז די געצייג זענען שוין דא אויפן מארק. מען דארף זיי באנוצן און מיט זיי עקספערימענטירן. ביז מען פרובירט נישט, קען מען נישט מאכן קיין ריכטיגע באשלוס.

פרייזן, טאקען און די צוקונפט (7:37:33)

דער קורס ווארנט אז אלע די געצייג נעמען געלט. אין א געוויסן פונקט שטעלן זיי א טאג-לימיט אדער א טאקען-לימיט. דאס הייסט, יעדן טאג קען מען אויסגעבן נאר א געוויסע צאל טאקענס.

מיט דער צייט, לויט דעם וואס טעכנאלאגיע-ליבהאבער פאראויסזאגן, וועלן די פירמעס אראפנעמען די פרייזן. דאס וועט געשען ווייל די געצייג וועלן ווערן א טייל פונעם טאג-טעגלעכן לעבן אין די קומענדיגע חדשים אדער יארן. די קאסטן וועלן ווערן מינימאל, און די געצייג וועלן אפשר ווערן אומזיסט.

AI געצייג פאר קאוד: Copilot און אנדערע (7:38:04)

דערנאך גייט דער קורס איבער צו די בעסטע גענעראטיוו AI געצייג, און ווי אזוי זיי טוישן פארשידענע אינדוסטריעס. ער הייבט אן מיט קאוד.

דער ערשטער איז Copilot . ער ווערט באשריבן ווי אן אויסגעצייכנטער געצייג, און רוב פירמעס באנוצן אים. Copilot ווערט אריינגעבויט אין די IDE-ס, די פראגראמען ווו מען שרייבט קאוד.

אזוי ארבעט עס:

- דער פראגראמירער האט א פראבלעם צו לייזן, למשל ער בויט אן אפליקאציע אדער שרייבט א לאגיק.
- ער באשרייבט קלאר די לאגיק.
- Copilot שרייבט דעם קאוד פאר אים.
- מען קען גלייך לויפן דעם קאוד.

דער קורס זאגט אז דאס האט געשפארט א סך צייט. אויסער דעם ווייזט Copilot נייע מוסטערן ווו דער פראגראמירער האט פריער נישט געקענט.

עס ווערט ערקלערט פארוואס עס הייסט "קא-פילאט". דער פראגראמירער פירט, און ער שרייבט די לאגיק. דער געצייג פארריכט אים, פארשטייט אים, און גיט אים דעם רעזולטאט.

נאך געצייג פאר קאוד וואס פירמעס באנוצן: Codesquire.ai און Mutable AI. מען קען זיי נאכקוקן און באנוצן.

שמועס-געצייג, פלאו-געצייג און אינטעגראציע (7:39:35)

פאר שמועסן און טעקסט ווערן אויסגערעכנט: Claude AI, Chatsonic, PaLM 2, ChatGPT, Gemini, Grok און Notion AI. מיט די אלע מאכט מען א שמועס מיט דעם מאדעל. ChatGPT און Grok האט מען שוין פריער באשפראכן. Jasper ווערט אויך באצייכנט ווי זייער גוט.

פאר "פלאוס", דאס הייסט ארבעטס-פראצעסן, ווערן דערמאנט נעמען ווו הערן זיך ווי Fireflies, Tome און Bardeen, און אויך Auto AI. די נעמען זענען נישט קלאר אין דער רעקארדירונג.

דער קורס ערקלערט אז יעדע פירמע אנטוויקלט אזעלכע GPT-ס לויט א ספעציפישן ביזנעס-באדערפעניש אדער ארבעטס-פלאו.

עס ווערט ווידער באטאנט אז יעדע ארגאניזאציע נוצט היינט Copilot. ער ווערט אריינגעבויט אין VS Code און אין PyCharm, און אפילו אין Teams, ווי מען האט פריער דערמאנט. Copilot פארבינדט זיך מיט אנדערע אפליקאציעס.

דער קורס זאגט צו אז אין צווייטן טייל וועט מען ווייזן א דעמא פון א ChatGPT געצייג וואס מען קען פארבינדן מיט דעם בראווזער.

געצייג פאר דיזיין, קלאנג און ווידעא (7:41:11)

פאר דיזיין ווערן אויסגערעכנט:

- E·DALL – א זייער עפעקטיוועך געצייג פון OpenAI.
- Canva.
- Designs.ai.
- Logo AI.
- Midjourney.
- Stable Diffusion – ווערט אויך באצייכנט ווי זייער גוט.

מיט די געצייג שאפט מען קרעאטיווע דיזיינס און בילדער, דורכדעם וואס מען גיט זיי א באשרייבונג.

פאר קלאנג ווערן דערמאנט Audiocraft און ElevenLabs.

פאר ווידעא ווערן דערמאנט Synthesia, און נאך נעמען וואס זענען נישט קלאר אין דער רעקארדירונג.

א וויכטיגע וואַרנונג: טייל געצייג זענען אומזיסט, טייל קאסטן געלט, און טייל געבן א חודש אבאנעמענט. אבער צום סוף זענען זיי אלע קאמערציעל, און יעדער פארלאנגט מען זאל באצאלן. מען קלייבט דעם געצייג וואס פאסט זיך דאס בעסטע פאר דער אייגענער פראדוקטיוויטעט.

פערזענלעכע איבערלעבונגען און עצות (7:42:42)

אין דער לעקציע ווערט דערציילט וועלכע געצייג מען האט פערזענלעך באנוצט. Copilot באנוצט מען שוין א לאנגע צייט פאר קאוד, און ער האט זיך באוויזן ווי עפעקטיוו. Grok, ChatGPT און Jasper באנוצט מען אויף זייער אומזיסטן צוטריט.

עס ווערט דערמאנט אז מען קען באקומען נאך אומזיסטן צוטריט דורך טוישן די אימעיל.

די עצה איז אזוי: מען באקומט א חודש אבאנעמענט און מען פרובירט אויס דעם געצייג. אויב מען זעט אז מען קען נישט אויסקומען אן אים, דעמאלט באצאלט מען און מען באנוצט אים ווייטער.

דעמא: צופאסן א רעזומע צו א דזשאב מיט ChatGPT (7:43:12)

איצט הייבט זיך אן א פראקטישע דעמא: ווי צו פאסן א רעזומע און א קאווער-לעטער מיט ChatGPT.

די פראבלעם איז אזוי. מען האט א רעזומע, אבער מען איז נישט זיכער צי עס פאסט צו דער באשרייבונג פונעם דזשאב (job description). פארשידענע פירמעס פארלאנגען פארשידענע זאכן. אויב דער רעזומע ערפילט

נישט די קריטעריעס פון דער פירמע, ווערט מען נישט אויסגעקליבן פאר דער קורצער ליסטע (shortlisting). מיט ChatGPT קען מען שנעל און עפעקטיוו צופאסן דעם רעזומע צו דער דזשאב-באשרייבונג, און שפארן צייט.

פאר דער דעמא ווייזט מען די ChatGPT אפליקאציע אין דער אומזיסטער ווערסיע. מען קען דריקן אויף "Upgrade" כדי צו באקומען דעם פלאן פאר 20 דאלאר א חודש.

דער פלאן גיט:

- Deep Research.
- כמה ריזאנירונג-מאדעלן: o4-mini, o4-mini-high און o3.
- א פארויסקוק אויף GPT-4.5.
- באגרעניצטן צוטריט צו Sora, פאר מאכן ווידעאס.

דער קורס ערקלערט אז ChatGPT האט אונטער-געצייג און אונטער-אפליקאציעס פאר ספעציפישע געביטן, למשל Canva אדער שרייבן טעקסט. אויב מען שפירט אז מען דארף דאס פאר דער טאג-טעגלעכער ארבעט, קען מען אפגרעידן.

דער פראמפט: קאנדידאט-פראפיל און אנווייזונגען (7:45:44)

נאכדעם ווערט ערקלערט דער פראמפט וואס מען האט אריינגעשיקט. ער באשטייט פון צוויי טיילן.

דער ערשטער טייל איז די אינפארמאציע וועגן דעם קאנדידאט:

- נאמען: Alex Johnson.
- דערפארונג: פינף יאר.
- אינדוסטריע: אינפארמאציע-טעכנאלאגיע.
- א רשימה פון זיינע סקילס.
- פאזיציע: Full Stack Developer ביי דער פירמע Innovative Tech Solutions אין California.
- א באשרייבונג פון זיינע ארבעטס-אחריותן.

דער צווייטער טייל זענען די אנווייזונגען צום מאדעל. מען בעט אים שאפן א רעזומע פאר Alex וואס זאל ווייזן זיין פינף-יאריגן וועג אין IT אלס full stack developer.

דער רעזומע זאל הויכהייבן:

- זיין קענטעניש אין Python, Java און JavaScript.
- די פריימווערקס Django און React.
- זיינע סקילס אין דאטאבאזעס און קלאוד-טעכנאלאגיעס, ווי PostgreSQL, MongoDB און AWS.
- זיין פירערשאפט אין פראיעקט-אנטוויקלונג.
- זיין צוואמענארבעט מיט א טים.
- זיין ליַדנשאפט פאר טעכנאלאגיע און זיין אנגייענדיגע לערנען.

דער קורס באטאנט אז "Alex" קען זיין דער לייענער אליין. אין ערשטן טייל גיט מען דעם רעזומע פון א מענטש. אינעם צווייטן טייל זאגט מען צום לארדזש-לענגוויטש-מאדעל וואס צו טון דערמיט.

א טיפ: אויב א פירמע האט ספעציפישע דזשאב-פארלאנגען, קען מען זיי אויך צולייגן אינעם פראמפט. מען פרעגט דעם מאדעל ער זאל מאדיפיצירן דעם אויבנדערמאנטן רעזומע אז עס זאל פאסן צו יענע פארלאנגען. דאס איז שרייבן א געהעריגן פראמפט.

דער רעזולטאט און וואס מען לערנט דערפון (7:47:50)

מען דריקט Enter, און ChatGPT שאפט א רעזומע.

אויבן שטייט דער נאמען Alex, מיט א שטאט וואס הערט זיך ווי San Francisco . דערנאך קומען אן אימעיל, א LinkedIn לינק, א GitHub לינק, און א "Professional Summary".

דער קורס באמערקט אז דער אריינגעשיקטער טעקסט איז נישט געווען אזוי סומירט און ארגאניזירט. איצט זעט עס אויס ווי א פראפעסיאנעלער רעזומע.

עס ווערט ערקלערט ווי אזוי אן HR מענטש אדער א רעקרוטער וועט קוקן אויף דעם. אין דער פראפעסיאנעלער סומע זעט ער די טעכנאלאגיעס Python, Java און JavaScript. ער קען זיי גרינג "פארסן", דאס הייסט אויסזוכן, און פארגלייכן מיט דעם וואס זיין פירמע פארלאנגט.

דער רעזומע אנטהאלט די פאלגנדע אפטיילונגען:

- Core Competencies – מיט React, Node.js, Python, full stack, און אויך Jenkins.
- Professional Experience – וועלכע פראיעקטן מען האט געטון און וואס מען האט דארט געארבעט. דאס איז געזומען פונעם ארגינעלן רעזומע אליין.
- Education און Certifications.
- Project Highlights.
- Soft Skills און טעכנאלאגיעס.

צום סוף פארלייגט ChatGPT נאך הילף. ער קען צוגרייטן א Word אדער PDF ווערסיע, אדער שרייבן א צוגעפאסטן קאווער-לעטער פאר דער פאזיציע.

דער קורס פאסט צוואם דעם פרינציפ: מען האט אריינגעגעבן געוויסע דאטא, און אויף דעם יסוד פון יענער דאטא בעט מען עפעס. דער פראמפט באשטייט פון דאטא און פון אנווייזונגען (instructions).

וואס איז א פראמפט און פון וואס באשטייט ער (7:49:54)

דער רעזומע-ביישפיל: קאנטעקסט און אנווייזונגען (7:49:54)

דער קורס פירט ווייטער דעם ביישפיל פון פריער, וואו מען האט געגעבן א רעזומע צום LLM. דער אויבערשטער טייל, וואס מען האט אריינגעשטעלט, איז דער רעזומע אליין, און ער דינט אלס קאנטעקסט. מיט דעם קאנטעקסט, צוזאמען מיט זיין אלגעמיינע וויסן, דארף דער מאדעל ארויסגעבן א בעסערן רעזומע: מער אויסגעפיילט, מער אטראקטיוו און בעסער ווי דער אריגינעל.

אינעם פראמפט האט מען אויסדריקליך געהייסן אז דער נייער רעזומע זאל אונטערשטרייכן געוויסע פונקטן: פירערשאפט אין פראדוקט-אנטוויקלונג, צווזאמענארבעט אין א טיעם, און ליבשאפט צו טעכנאלאגיע.

דער קורס שלאגט פאר אז יעדער וואס האט זיין אייגענעם רעזומע זאל אים אריינשטעלן אנשטאט דעם ביישפיל און דערמיט פראבירן. מען קען אויך בעטן אז דער רעזולטאט זאל קומען פארמאטירט, למשל אלס PDF אדער אין Word.

א ווארענונג: מען זאל אכטונג געבן אז דער אקאונט זאל זיך נישט אויסשעפן. סיי דער טעקסט וואס מען שיקט אריין און סיי דער רעזולטאט וואס ווערט ארויסגעגעבן ווערן געראכנט אין טאָקענס, און ס'קען זיין א טעגלעכע גרענעץ. פאר געוועניליכע טאג-טעגליכע ארבעט איז עס אבער גאנץ ניצליך.

ווי אזוי מעסט מען די קוואליטעט פון LLM-ס (7:51:28)

עס ווערט ערקלערט אז ס'איז נישטא קיין פעסטער כלל וועלכן LLM מען דארף ניצן; עס קען זיין ChatGPT, Gemini אדער אן אנדערער. די שווערע פראגע איז ווי אזוי מען אפשאצט די סיסטעמען: צי זיי געבן א ריכטיגן ענטפער לויט דער דאטא, דער פראגע אדער דעם פראמפט וואס מען האט געגעבן.

א ביישפיל: מען שרייבט א פראבלעם און בעט א קאוד. דער מאדעל גיט א קאוד וואס זעט אויס פונקט ווי געבעטן, אבער ער לויפט נישט ריכטיג און גיט פעלערן. אמאל ארבעט א מאדעל גוט ביי איין אויפגאבע און פאלט שטארק דורך ביי אן אנדערער. מען קען דעמאלט פראבירן מיט אנדערע מאדעלן און פארגלייכן די פינקטלעכקייט, אבער קיין פעסטער כלל פאר אפשאצונג איז נישטא.

פונדעסטוועגן זענען דא געוויסע מעסטן:

- Perplexity: עס ווערט באטאנט אז דאס איז נישט קיין געצייג, נאר אן אלגעמיינער באגריף, א מאס צו אפשאצן א שפראך-מאדעל. דערמיט קען מען זען וועלכער מאדעל באקומט א בעסערע נאטע. למשל, ווען מען פרעגט די זעלבע פראגע ביי ChatGPT, Gemini, Falcon אדער Claude, קען מען זען ווער ארבעט בעסער.
- מענטשליכע אפשאצונג (human evaluation): דער באניצער אליין ווייסט בעסטנס צי דער ענטפער פאסט פאר זיין פראבלעם און וועלכער ארבעט בעסער. אין דעם שטח זענען מענטשליכע אפשאצער די בעסטע.

וואס איז א פראמפט? (7:54:04)

דער קורס פרעגט די צוהערער וואס א פראמפט איז, און די ענטפערס זענען געווען: "דער אינפוט וואס מיר געבן", "א באפעל" און "א מעסעדזש". דערנאך קומט די דעפיניציע: א פראמפט איז טעקסט אין נאטירליכער שפראך וואס הייסט א גענעראטיוון AI מאדעל דורכפירן א געוויסע אויפגאבע.

דער רעזומע-ביישפיל ווייזט דאס קלאר: ערשט האט מען געגעבן קאנטעקסט, דעם רעזומע, און דערנאך א סעט אנווייזונגען ווי אזוי דער רעזומע זאל געטוישט ווערן.

א גוט געבויטער (well-crafted) פראמפט פירט דעם GenAI צו פארשטיין דעם קאנטעקסט און די פונקטליכע פאדערונגען פון דער אויפגאבע. וואס הייסט "גוט געבויט"?

- ער האט ריכטיגע אפשטאנדן און ריכטיג געגעבענע אנווייזונגען.
- ער איז קלאר און פרעציז.

- ער איז נישט אומקלאר אדער אומפולשטענדיג. מען זאל נישט מיינען אז דער מאדעל וועט שוין אליין כאפן וואס מען מיינט. אמאל קען ער עס דערקענען, אבער די שאנסן זענען קליין. א פראמפט מיט ריכטיגע אנווייזונגען וועט געבן דעם בעסטן ענטפער.

קאסטן, טאָקענס און דער פראמפט אלס בריק (7:56:06)

דער צווייטער פונקט איז די קאסטן. ווי געזאגט, ווערט אלעס גערעכנט אין טאָקענס, און איינער פון די לערנער האט שוין אויסגעשעפט זיין טעגלעכע גרענעץ. דעריבער דארף מען זיין שפארעוודיג און ניצן די טאָקענס אזוי אז זיי זאלן קלאר באשרייבן די אויפגאבע.

א פרעציזער פראמפט פירט צו א מער פאקוסירטן און ניצליכן רעזולטאט. ער דינט אלס בריק צווישן דער כוונה פון דעם באניצער און דער דורכפירונג פון דעם AI: וואס איך וויל, און ווי אזוי דער AI פירט עס אויס.

דער קורס פאסט עס צוזאמען: א פראמפט איז א סעט אנווייזונגען צום LLM כדי ער זאל טון די געוואונטשענע ארבעט. ער דארף זיין קלאר, פרעציז און גוט געבויט. מען זאל נישט אריינלייגן איבעריגע אפשטאנדן, ווייל א איבעריגער אפשטאנד ווערט אויך געראכנט אלס א טאָקען. דער מאדעל דערקענט אויך מוסטערן אין דעם וואס מען האט געשריבן, און לויט דעם מוסטער פרובירט ער פארשטיין די שפראך און געבן דעם פונקטליכן ענטפער.

פשוטע און קאמפליצירטע פראמפטס (7:57:37)

א פראמפט קען זיין א פשוטע, גלייכע פראגע, און דער מאדעל ענטפערט לויט זיין טריינירונג-דאטא. ביישפיל: "What is the capital of France". דער מאדעל דערקענט די שליסל-ווערטער, ווי "France" און "capital", און לויט דעם גיט ער דעם ענטפער.

פראמפטס קענען אבער זיין פיל מער קאמפליצירט און דינען פאר פארשידענע אויפגאבעס: פארענדיגן טעקסט, איבערזעצונג, צוזאמענפאסונג און אפילו שעפערישע שרייבעריי. דער רעזומע-פראמפט איז א ביישפיל דערפון: מען האט געגעבן אינפארמאציע און געבעטן מען זאל זי פארבעסערן.

וויפל דער ענטפער טויג, העננט אפ ווי גוט דער פראמפט איז געבויט. מען זאל זיך אבער נישט וואונדערן אז דער זעלבער פראמפט גיט פארשידענע רעזולטאטן ביי פארשידענע באניצער. דאס זענען פראבאביליסטישע מאדעלן: ווען מען שיקט אריין די זעלבע אנווייזונגען עטליכע מאל, באקומט מען נישט שטענדיג דאס זעלבע, און אמאל קען דער מאדעל אויך "האלוצינירן", דאס הייסט אויסטראכטן אינפארמאציע.

די חלקים פון א פראמפט (7:59:40)

דער קורס דערקלערט וואס מען דארף אריינלייגן אין א פראמפט, מיט א געדאכטן ביישפיל. ערשטנס זאגט מען דעם LLM אלס וועמען ער זאל זיך פירן: "דו ביסט א מעדיצינישער פראקטיקער", אדער א מעדיצין-סטודענט, א דאקטער, א ריכטער אדער אן אדוואקאט. דאס הייסט דער "user set": מען הייסט דעם מאדעל שפילן א געוויסע ראלע לויט דעם פאך אדער לויט דעם באדערפעניש.

דערנאך גיט מען אים כללים וואס ער דארף פאלגן, און אינפארמאציע: למשל טעקסט, בילדער אדער דאטא פון א פאציענט. צום סוף זאגט מען אים וואס פאר א רעזולטאט מען וויל. אזוי באשטייט א פראמפט פון:

- דער באניצער אדער די ראלע
- אנווייזונגען

- קאנטעקסט
- דער רעזולטאט (output)

מען קען בויען אזא פראמפט לויט יעדן באדערפעניש. אינעם רעזומע-פראמפט איז דער אינפוט, דער טעקסט פון דעם רעזומע, געווען דער קאנטעקסט, און דערנאך זענען געקומען די אנווייזונגען פאר דער אויפגאבע. דארט האט מען אבער ביידע צוזאמענגעשמאלצן.

מען קען עס צעטיילן אין פשוטערע טריט. די אנווייזונגען קען מען שרייבן אלס באזונדערע פונקטן, למשל:

- גיב עס איבער אין צוויי שורות
- אין Word פארמאט
- אלס JSON דאקומענט
- נישט מער ווי 200 אותיות

און באזונדער פון דעם זאגט מען די אויפגאבע אליין: ארבעט לויט די אנווייזונגען און מאך דעם רעזולטאט. ביים ביישפיל האט מען עס פאַרפשוט'ט און אלעס געגעבן אין איין אויפגאבע, אבער ווען די לאגע איז קאמפליצירט אדער דער פראבלעם איז גרויס, איז כדאי צו צעטיילן.

ביישפילן פון פארשידענע אינדוסטריעס (8:02:50)

אין דער געזעץ-וועלט: מען זאגט דעם מאדעל "דו ביסט א ריכטער", און גיט אים אלס אינפוט א קעיס-פייל, א טעקע פון הויף-שריפטן אדער PDF דאקומענטן, וואס דינט אלס קאנטעקסט. דערנאך גיט מען אנווייזונגען וואס צו טון מיט דעם קעיס, למשל: "ניץ דעם קאנטעקסט און ציה מיר ארויס די ערשטע צד, די צווייטע צד, דעם נאמען פונעם ריכטער" און אנדערע פרטים.

דער מעדיצינישער ביישפיל איז שוין דערמאנט געווארן פריער. אן אנדערע אינדוסטריע איז E-commerce. דארט קען די אינפוט-דאטא זיין אין JSON, אין Word, אדער פון א דרויסנדיגן וועבזייטל וואס גיט דעם קאנטעקסט. עס קענען אויך זיין בילדער, א דאטא-סעט אדער די אויפגאבע אליין. לויט דעם קען מען בעטן: "גיי צו יענעם זייטל, פיר דורך די ארבעט, און פון די בילדער פארגלייך מיר די באווערטונגען (ratings) אדער די פרייזן", אאז"וו, לויט וואס דער באדערפעניש פאדערט. דאס זענען די חלקים פון א באזיס-פראמפט.

פראמפט-פריימווערקס און פראקטישע איבונגען (8:04:57)

פראמפט-פריימווערקס: ReAct און Chain of Thought (8:04:57)

דער קורס ענדיגט דעם ענין פון די פארשידענע טיילן וואס א פראמפט דארף אנטהאלטן, און גייט איבער צו נאך א וויכטיגן פונקט: עס זענען פאראן אזוי געראפענע פראמפט-פריימווערקס, ד.ה. באשטימטע מוסטערן ווי אזוי מען שטעלט צוזאם א פראמפט. צוויי פון די מערסט פארשפרייטע פריימווערקס ווערן דערמאנט: ReAct און Chain of Thought, און עס זענען פאראן נאך.

עס ווערט ערקלערט וואס Chain of Thought מיינט: מען גיט דעם LLM געוויסע אנווייזונגען, און מען זאגט אים: טראכט אריבער איבער די אנווייזונג אדער איבער דעם סטעיטמענט, און גיב מיר א שריט-נאך-שריט אנאליז. מען קען פשוט בעטן אז נאכן טראכטן זאל ער אנווענדן א שריט-ביי-שריט אנאליז און ערשט דערנאך געבן דעם ענטפער. אין אזא פריימווערק האט מען א באשטימטע אויפגאבע, א מסקנא וואס מען וויל דערגרייכן, און א רעזולטאט (output) וואס מען וויל באקומען.

די הויפט-טיילן פון א פראמפט און די פארעם (8:06:31)

דער קורס חזר'ט איבער די טיילן וואס זענען שוין פריער קלאר געמאכט געווארן: עס איז דא א באנ׳יצער, מען גיט א קאנטעקסט, דערנאך א אינפוט, דערנאך אנווייזונגען און די אויפגאבע. א צווייטער אופן איז צו פרעגן: וואס איז די אויפגאבע, וואס איז די מסקנא, און וואס איז דער רעזולטאט – אלעס באזירט אויף א קאנטעקסט. דאס זענען די וויכטיגסטע באשטאנדטיילן ביים באנוצן א פראמפט.

עטליכע עצות ווערן געגעבן:

- דער פראמפט דארף זיין געניו און קלאר.
- ער דארף האבן ריכטיגע אפשטאנדן (spacing) – אויך דאס שפילט א ראלע.
- עס איז גוט צו געבן די אנווייזונגען אין JSON, אין סטרוקטורירטן טעקסט אדער אין XML.

דער קורס ערקלערט פארוואס: עס איז באמערקט געווארן אין געוויסע וויסנשאפטלעכע ארבעטן אז JSON און XML זענען גוט סטרוקטורירטע שפראכן, און ווען מען שפייזט אריין די דאטא אויף אזא אופן, איז דער שאנס פאר טעותים קלענער און דאס פארשטיין בעסער. למשל, אנשטאט צו שרייבן פשוטן טעקסט, קען מען געבן "name" מיט'ן ווערט "John", און דערנאך אנווייזונגען, טעקסט, בילדער אדער די קעפלעך פון א רעזומע. אנשטאט באלעטן-פונקטן קען מען זיי אוועקלאזן און אלעס געבן אלס JSON-סטרוקטור. דאס פארמינערט אויך די קאסטן, און עס איז אן עפעקטיוואר אופן ווי צו געבן א פראמפט פאר א LLM, ווייל דאס איז א סארט "query"-שפראך, און די גרויסע שפראך-מאדעלן ארבעטן אליין אויף אזא שפראך. ווען מען טראכט ווי א פראגראמירער, באקומט מען די מערסט פאסיגע אנווייזונגען און רעזולטאטן.

א ביישפיל: צעטיילן א פראמפט אויף זיינע טיילן (8:08:03)

דער קורס נעמט א ביישפיל-פראמפט און צעטיילט אים לויט די פיר טיילן – אינפוט, קאנטעקסט, אנווייזונגען און אויפגאבע. דער פראמפט בעט צו שאפן א קורצע דערציילונג וועגן א "wizard quest" (די זוכענ׳ש פון א צויבערער) אין א מאגישן וואלד, מיט א איבערראשנדן סוף, און צו צושטעלן באשרייבונגען פון באשעפענישן.

- "שאף א קורצע דערציילונג" – דאס איז די אויפגאבע וואס מען וויל אויספירן.
- "וועגן א wizard quest" – דאס איז אן אנווייזונג.
- "אין א מאגישן וואלד" – דאס איז דער קאנטעקסט, ד.ה. וואו די געשיכטע קומט פאר.
- "א איבערראשנדער סוף" – דאס איז אויך אן אנווייזונג.
- "באשרייבונגען פון באשעפענישן" – דאס איז דער אינפוט וואס מען האט געגעבן.

אלע די זאכן קען מען אויך געבן אין JSON-פארמאט: א פעלד פאר task, א פעלד פאר instruction, א פעלד פאר context און א פעלד פאר input.

דערצו קען מען אויך באשטימען א ראלע: "שפיל די ראלע פון א ריכטער", "פון א דאקטער", "פון א סעילס-אגענט", "פון א CEO פון דער פירמע", "פון א דאטאבעיס-אינזשעניר" און אזוי ווייטער. דער מאדעל וועט זיך דאן פירן לויט דער ראלע און ארבעטן בעסער.

דער קורס פאסט עס צוזאם: אזוי צעברעכט מען די אויפגאבע אויף פשוטע און לייכטע שריט. עס איז ווי מען ערקלערט עפעס פאר א קינד פון צוויי יאר – מען לייגט אים אריין מיט א לעפל, מען גיט אים אלעס גענוי, מיט קורצע און באשטימטע אינפארמאציע. דאס איז דער גאנצער עיקר פון פראמפט-דיזיין.

א קורצע פרובע-פראגע (8:10:39)

עס ווערט געשטעלט א פראגע: וועלכער באשטאנדטייל דארף זיין אין א פראמפט פאר א שפראך-מאדעל צו איבערזעצן א טעכנישן דאקומענט פון ענגליש אויף פראנצויזיש? די ברירות:

- A – באשטימען די מקור-שפראך און די ציל-שפראך אין די אנווייזונגען.
- B – באנוצן א דאטא-סעט וואס האט נישט צו טון מיט דעם טעכנישן פעלד פון דעם דאקומענט.
- C – נישט אריינרעכענען קיין הינטערגרונט-אינפארמאציע וועגן דעם אינהאלט.
- D – אריינלייגן טעכנישע אויסדרוקן אן ערקלערונג.

דער ריכטיגער ענטפער איז A: מען מוז באשטימען די מקור- און ציל-שפראך – דאס איז דער גרונט-באדינג.

פראקטיש: איבערזעצן א טעקסט פון א בוך (8:11:43)

דערנאך גייט דער קורס איבער צו פראקטישע איבונגען פון טאג-טעגלעכע באנוצונגען: שפראך-איבערזעצונג, שאפן א LinkedIn-פראפיל, און סענטימענט-אנאליז (אויב עס בלייבן נאך טאָקענס).

דער סצענאר: מען פירט אן אנליין ביכער-געשעפט וואס פארקויפט ביכער אין פארשידענע שפראכן, און מען וויל באנוצן ChatGPT פאר איבערזעצונג אין רעאלער צייט. א קונה זעט א בוך און וויל א טעקסט דערפון איבערגעזעצט אויף א צווייטער שפראך – שפאניש, דייטש, ענגליש, הינדי אדער אנדערע. מען וויל זען צי וועלן מען גיט דעם טעקסט צו GPT מיט א סעט אנווייזונגען, וועט ער עס איבערזעצן. דאס פאסט מיט דעם וואס מען האט געלערנט אין דער טעאריע, אז ChatGPT קען דינען פאר איבערזעצונג; פריער האט מען שוין געזען טעקסט-שאפונג און א רעזומע.

דער ערשטער פראמפט האט אנגעהויבן מיט א באשרייבונג פון דעם סצענאר (אן אנליין ביכער-געשעפט מיט טיטלען אין פילע שפראכן פאר א וועלטלעכן עולם). דער קורס ווייזט אז דער סצענאר איז נישט נייטיג; מען קען אים אוועקנעמען און שרייבן גלייך: "Translate the following book excerpt into multiple languages to make it accessible to a broader audience", מיט'ן טעקסט נאכדעם. וועט מען לויפט עס, זעט מען איבערזעצונגען אויף הינדי, שפאניש, פראנצויזיש, כינעזיש, אראביש, דייטש און רוסיש.

פראקטיש: שאפן אן איינדרוקספולן LinkedIn-פראפיל (8:14:27)

די צווייטע איבונג: שאפן אן איינדרוקספולן LinkedIn-פראפיל מיט ChatGPT, מיט א ציל וואס איז מער ווי בלויז א פערזענלעכער "בראנד". עס ווערט געגעבן א מוסטער-פראמפט וואס יעדער קען ענדערן, ווייל יעדער מענטש האט אן אנדער פראפיל; לויט דעם אייגענעם פראפיל קען מען געבן אנווייזונגען און פרובירן שעפערישע וועגן אז דער פראפיל זאל זיך אריסהייבן.

מען זעט אז דער מאדעל האט אויך געגעבן א מעסעדזש צו אנהייבן א שמועס, ווי: "היי... מיר קענען אויסטוישן איינזיכטן אדער אפילו מיטארבעטן אויף א פראיעקט, איך קוק ארויס צו פארבינדן זיך". אזוי קען מען טון נעטווארקינג דורך אימעיל אדער LinkedIn-מעסעדזשעס. ער האט אויך געשאפן א LinkedIn -פאוסט מיט SEO-השטעגס. דאס איז געווען באזירט אויף א רעזומע – נישט די אייגענע, נאר איינע ארונטערגעלאדן פון אינטערנעט; ענליך קען יעדער עס טון מיט זיין אייגענער רעזומע.

דער פראמפט האט געבעטן: אין דער "experience"-אפטיילונג, גיב מיר דעטאלירטע איינטראגן וואס ווייזן מיינע ביישטייערונגען און דערגרייכונגען, מיט טעטיקייט-ווערבן און מיט צאל-מעסיגע רעזולטאטן; אויך

פארשלאג אינהאלט צו אנהייבן שמועסן מיט נייע פארבינדונגען, וואס קאנצענטרירט זיך אויף לעצטע נייעס אין דער אינדוסטריע – ביז 100 ווערטער.

דער קורס באטאנט פארוואס מען באגרעניצט די ווערטער: אנדערש ווען דער מאדעל ווייטער און ווייטער שרייבן ביז'ן מאקסימום טאָקענס; מען קען באגרעניצן ביז וויפיל ווערטער. מען קען אויך ארויפלאדן א דאקומענט און דערפון שאפן LinkedIn-פראפילן, קעפלעך (headlines), אימעילס אדער שיינע טעקסטן. אין עטליכע סעקונדעס האט מען באקומען אנווייזונגען וואס מען קען גלייך ארויפלייגן אויפן LinkedIn-פראפיל. די איבונג קען מען אויך שפעטער טון.

פראקטיש: סענטימענט-אנאליז (8:18:23)

די דריטע אויפגאבע איז סענטימענט-אנאליז (נאך טעקסט-שאפונג מיט דער רעזומע און דעם LinkedIn-פראפיל). מען גיט דעם מאדעל אנווייזונגען ווי אזוי צו באצייכענען סענטימענטן – פאזיטיוו, נעגאטיוו אדער נייטראל – לויט דעם קאנטעקסט פון דעם אינפוט-טעקסט.

אין דער דעמאנסטראציע גיט מען בלויז איין רעצענזיע (review), כאטש מען קען געבן מער; מען האט זיך באגרעניצט. דער פראמפט זאגט: קאטעגאריזיר די רעצענזיע לויט: positive, negative, neutral, mixed, question אדער appreciative, און דערנאך: "provide the sentiment for the following reviews" – מיט איין רעצענזיע, און מען קען לייגן צוויי, דריי, פיר. מען האט צוגעגעבן אנווייזונגען, און דערין קען מען אריינלייגן שליסל-ווערטער, למשל אז "good", "happy" באצייכענען פאזיטיווע באמערקונגען, און ענליך פאר נעגאטיווע.

מולטימאדאל AI און די געשיכטע פון LLMs (8:20:26)

ענדיגן דעם סענטימענט-אנאליז ביישפיל (8:20:26)

דער קורס ענדיגט דעם פריערדיגן ביישפיל פון סענטימענט-אנאליז. עס ווערט ערקלערט אז אין פראמפט קען מען צולייגן געוויסע שליסל-ווערטער אדער אנווייזונגען, כדי דער מאדעל זאל וויסן ווי צו אפשאצן דעם טעקסט. למשל קען מען שרייבן אז א נעגאציע, א "ניין", אדער סיי וועלכע מחלוקת אין טעקסט באדייט א נעגאטיווע מיינונג. אויף דעם יסוד קען מען בעטן דעם מאדעל: "לייען דעם טעקסט און גיב מיר דעם סענטימענט."

מען זעט דעם רעזולטאט: דער מאדעל האט געענטפערט אז דער שרייבער פון דער רעצענזיע האט אויסגעדריקט סיי פאזיטיווע און סיי נעגאטיווע מיינונגען. ער האט אויסגערעכנט די נעגאטיווע פונקטן, און צום סוף גענעבן א "געמישטן סענטימענט" (mixed sentiment). דער קורס מוטיגט מען זאל אליין פרובירן אנדערע אפציעס, און באזונדערע טעקסטן ווערן צוגעשיקט צום פראקטיצירן.

וואס הייסט "מולטימאדאל"? (8:21:16)

דערנאך הייבט אן א נייע טעמע: מולטימאדאל (multimodal). עס ווערט ערקלערט אז דער טערמין קומט פון דעם ווארט "מאדאליטעטן" (modalities), און נישט פון "מאדולן". מאדאליטעטן זענען די ווענן ווי מען באקומט אינפארמאציע. ביים מענטש זענען דאס די פינף חושים: אנריר, הערן, זען, פארזוכן, און שמעקן.

ביי AI און LLMs זענען דא בלויז טייל פון די מאדאליטעטן:

- טעקסט

- קלאנג (סאונד), וואס שטימט מיט הערן
- בילדער, וואס שטימט מיט זען (ראיה)

אנריר און פארזוכן ניצט מען דערווייל נישט, ווייל דאס איז, ווי עס ווערט געזאגט, "אויף אן אנדער מדרגה". הערן און זען ווערן יא געגנוצט. אזוי ווערט די ידיעה פון מענטשליכע מאדאליטעטן איבערגעטראגן אויף די מאדאליטעטן וואס א LLM קען פראצעסירן.

ביז אהער האט מען געזען נאר טעקסט: מען שרייבט א טעקסט, דער מאדעל באקומט עס ווי אינפוט און גיט צוריק א טעקסט ווי אויטפוט. מולטימאדאל מיינט אז דער אינפוט קען זיין טעקסט מיט א בילד, און דער אויטפוט קען זיין טעקסט אדער א בילד, לויט וואס מען בעט. דאס הייסן די מולטימאדאלע מעגליכקייטן פון א LLM.

ביישפילן פון מולטימאדאלע אנווענדונגען (8:23:34)

דער קורס ברענגט עטליכע ביישפילן:

- מען גיט א בילד מיט א פראמפט אין טעקסט, און מען בעט דעם מאדעל אויסצונוצן דאס בילד ווי קאנטעקסט און צו געבן א תשובה.
- מען גיט א קול-רעקארדירונג. דער מאדעל אנאליזירט דעם קלאנג און שרייבט א טעקסט: וועגן וואס מען רעדט, א סיכום, אדער א רעפערענץ.

עס ווערט ערקלערט אז די אינטעגראציע פון עטליכע מאדאליטעטן, טעקסט, בילדער, ווידעאס און אודיא, העלפט דעם מאדעל בעסער צו פארשטיין און צו שאפן תשובות. מען קען צווזאמשטעלן עטליכע אינפוטס, ווי טעקסט מיט א בילד. שפעטער ווערט מען זען א פראקטישע דעמאנסטראציע דערפון. דאס גיט א דינאמישע און אינטעראקטיווע תשובה.

ווייטערע ביישפילן:

- מעדיצינישע בילדער: מען גיט אריין א PET-סקען, MRI-סקען אדער X-ray, און מען פרעגט: "וואס מיינסטו לויט דעם סקען?" אזוי קען מען איבערקוקן צי דער דאקטער האט ריכטיג געמשפט.
- שטייער-פאפירן: מען גיט אריין דעם שטייער-פארם (Form 16 אין אינדיע) און מען בעט: "לויט דעם פארם, שטעל צווזאמען דעם בעסטן מעגליכן income tax return."
- א ביל אדער אן אינווּאיס (חשבון): דערמיט קען מען אויך טון אסאך זאכן.
- רייד אין א פרעמדער שפראך: אויב א רעקארדירונג איז נישט אויף ענגליש, נאר אויף שפאניש אדער יאפאניש, קען דער מאדעל העלפן פארשטיין דעם קאנטעקסט, וואס מען רעדט דארט.
- ווידעא-אנאליז: מען קען פארשטיין וואס א ווידעא וויל זאגן, און שרייבן א טעקסט-באשרייבונג פון דעם ווידעא'ס אינהאלט.

עס ווערט געזאגט אז GPT איז ארויס ווי א פירער אין דעם געביט. עס גיט מולטימאדאלע מעגליכקייטן וואס פאראייניגן נישט נאר טעקסט, נאר אויך בילדער, קלאנג און ווידעאס. דער קורס זאגט אן אז מען וועט באלד פרובירן פראקטיש ווי מען גיט אריין סיי בילד-אינפוט סיי טעקסט-אינפוט.

איבערחזרה: די אנטוויקלונג פון AI זינט 1950 (8:27:16)

פאר דעם פראקטישן טייל מאכט דער קורס א חזרה אויף וואס מען האט שוין געלערנט: די הקדמה צו געזעראטיוון AI (GenAI), און די אנטוויקלונג פון 1950 ביזן היינט.

דער Turing טעסט (1950ער יארן): עס ווערט ערקלערט אז דאס איז א טעסט צו זען צי א מאשין קען נאכמאכן א מענטש. עס איז דא א צימער מיט א מענטש. מען שאפט א קול, וואס קען זיין פון א מאשין אדער פון א מענטש, און דער מענטש וואס זיצט אינדרויסן דארף אויסגעפינען וואס עס איז. אויב ער דערקענט ריכטיג אז עס איז א מאשין, איז די מאשין דורכגעפאלן. אויב ער קען נישט דערקענען, זאגט מען אז די מאשין האט באשטאנען דעם טעסט.

דער ELIZA טשעטבאט: דערנאך ווערט דערמאנט דער טשעטבאט ELIZA, וואס איז געבויט געווארן מיט אנווייזונגען לויט שפראך-מוסטערן (regex patterns).

נויראל-נעצן, CNN און סעקווענצן (8:28:49)

דערנאך זענען געקומען נויראל-נעצן. לויט דעם קורס זענען זיי געווארן פאפולער אין די 1980ער יארן, און אלס גרויסער דורכברוך ווערט דערמאנט AlexNet. נויראל-נעצן האבן געהאלפן דערקענען בילדער און געפינען מוסטערן אין זיי.

מען ניצט דערפאר convolutional neural networks (CNN). וועז א בילד גייט אריין אין א convolution-שיכט, אנאליזירט זי די מוסטערן. דערפון שטעלט זי פעסט צי דאס בילד איז פון א הונט אדער פון א קאץ. אזוי קען מען קלאסיפיצירן אין פארשידענע קלאסן.

RNN, LSTM און GRU: עס ווערט ערקלערט אז דער גרויסער חסרון פון די פריערדיגע נויראל-נעצן איז געווען דער פיקסירטער לענג פון אינפוט. דעריבער האט מען נישט געקענט אריינגעבן אין א CNN סעקווענציעלע דאטא, ווי time series אדער NLP-טעקסט. פאר אזעלכע דאטא דארף מען אן ארכיטעקטור וואס נעמט א סעקוענץ פון דאטא אריין אין נויראל-נעץ און גיט ארויס א תשובה. למשל sequence-to-sequence, מאשין-איבערזעצונג, אדער איבערזעצן פון איין שפראך אין אן אנדערע. דערפאר זענען געקומען RNNs און LSTMs. GRU איז געווען נאך א מעכאניזם מיט "טויערן" (gated mechanisms), וואס האט אויך געהאלפן מיט סעקווענציעלע דאטא. אבער ביי RNNs זענען געווען שוועריגקייטן מיט gradient descent און אפטימיזאציע, וואס מען האט שפעטער באהאנדלט.

דער attention מעכאניזם (8:31:27)

דער גרויסער דורכברוך איז געקומען מיט דעם attention mechanism. עס ווערט ערקלערט אז דער ציל איז פעסטצושטעלן ווי וויכטיג א געוויס ווארט איז לגבי די אנדערע ווערטער אין זאץ. למשל אין א זאץ ווי "I want to go": ווי וויכטיג איז "I" אין פארהעלטעניש צו די אנדערע ווערטער? ווי וויכטיג איז "want"? אזוי גיט מען יעדן ווארט "אויפמערקזאמקייט", און מען באשליסט וועלכע ווערטער האבן א העכערע וויכטיגקייט.

פריער האבן אלע ווערטער אין סעקווענציעלע דאטא געהאט די זעלבע וואג, און עס איז נישט געווען קיין קאנטעקסט. יעצט קען מען זאגן אז "want" איז וויכטיג, "to" איז וויכטיג, "I" איז וויכטיג, אבער "go" איז נישט אזוי וויכטיג לגבי די אנדערע. דאס, זאגט דער קורס, איז דער עיקר פון attention: וועלכעס וואלט האט וויכטיגקייט לגבי די אנדערע ווערטער.

די Transformer ארכיטעקטור און LLMs (8:33:00)

דער attention מעכאניזם איז שפעטער אריינגעשטעלט געווארן אין דער Transformer ארכיטעקטור. זי האט אן encoder און א decoder, און ס'איז א קאמפליצירטע ארכיטעקטור וואס אנטהאלט:

- נויראל-נעץ שיכטן
- layer normalization
- skip connections
- multi-headed attention

דער אויטפוט פון encoder גייט אריבער צום decoder, וואס האט א masked attention, און פון דארט קומט דער סוף-אויטפוט. די ארכיטעקטור איז פארגעשלאגן געווארן אינעם ארטיקל "Attention Is All You Need". דער קורס רופט אים א state-of-the-art ארטיקל, וואס האט געוצט attention אין דער Transformer ארכיטעקטור און געלייזט די פראבלעמען פון sequence-to-sequence מאדעלירן מיט RNN.

מיט דער Transformer ארכיטעקטור זענען אויפגעקומען די LLMs. די GPT סעריע איז א decoder-only ארכיטעקטור פון א Transformer. עס ווערט אבער ערקלערט אז מען קען ניצן דעם encoder באזונדער, דעם decoder באזונדער, אדער ביידע צוזאמען. דאס זענען דריי סארטן ארכיטעקטורן: encoder, decoder, און ביידע. LLMs ניצן בדרך כלל דעם decoder, וואס איז נוצליך צו שאפן טעקסט.

דער קורס דערמאנט וואס מען האט שוין געזען אין פריערדיגע טיילן: מען האט געמאכט א סיכום פון א רעזומע, מען האט געזען סענטימענט-אנאליז, און קאוד-שאפונג. אט די טעכניקן זענען זייער נוצליך מיט דער GPT סעריע און LLMs. דאס איז די טעאריע הינטער ווי GPT איז אנטוויקלט געווארן.

חזרה: געזעראטיוער AI און LLMs (8:35:36)

דערנאך חזר'ט דער קורס איבער וואס איז געזעראטיוער AI: א קאטעגאריע פון AI-סיסטעמען וואס שאפן נייעם אינהאלט, ווי בילדער, טעקסט, מוזיק, אדער אנדערע קרעאטיווע פארמען. ער איז א טייל פון AI וואס שטיצט זיך אויף דיפ-לערנען און אויף דער Transformer ארכיטעקטור. זיינע צילן זענען:

- דערקענען מוזטערן
- פארלייגן א ברייטע אויסוואל פון מעגליכקייטן
- פארשטארקן די קרעאטיוויטעט
- דורכפירן עטליכע ארבעטן

עס ווערט געזאגט אז ער העלפט נישט נאר אין פראדוקטיוויטעט, נאר פארבעסערט אויך דאס טעגליכע לעבן. דער קורס מוטיגט אנצוהייבן ארבעטן מיט GenAI-געצייג.

LLM הייסט large language models. זיי זענען דער יסוד, און זיי העלפן שאפן טעקסט וואס זעט אויס ווי פון א מענטש, אין פארשידענע ארבעטן און געביטן. פארוואס "גרויס"? דאס ווארט "large" באציט זיך צו דער ריזיגער גרייס און קאמפליצירטקייט פון די מאדעלן. ווי שוין פריער געזאגט, האט GPT-3 בערך 175 ביליאן פאראמעטערס. פאראמעטערס זענען, ווי עס ווערט ערקלערט, די "ווייץ" (weights) פון וועלכע די LLMs זענען געבויט.

איבערבליק: LLMs, פראמפטן און מולטימאדאל (8:37:13)

פארוואס מען קען נישט טריינירן אן LLM אויף א היימישן קאמפיוטער (8:37:13)

דער קורס הייבט אן מיט א חזרה אויף די גרויסע שפראך-מאדעלן (LLMs). די מאדעלן זענען געטריינירט געווארן אויף א ריזיגע מאס דאטא, וואס מען מעסט אין ג'יגאבייטס, פעטאבייטס און טעראבייטס. דערפאר איז עס אוממעגלעך צו טריינירן אזא מאדעל אויף א געוועניליכן קאמפיוטער. מען דארף א גרויסע רעכן-קראפט, שטארקע הארדווער און שטארקע GPU- און TPU-מאשינען, און די מאשין דארף אויך האבן די פאסיגע פעאיקייט.

עס ווערט ערקלערט אז דער עיקר ציל פון די מאדעלן ליגט אינעם געביט פון) NLPנאטירלעכע שפראך-באארבעטונג). די דאטא קומט אריין ווי טעקסט. דער מאדעל וואס מען ניצט איז א Transformer-מאדעל מיט א decoder-ארכיטעקטור, וואס העלפט צו שאפן א סעריע ווערטער איינס נאכן אנדערן. ביישפילן פון ברייט-געוצטע LLMs זענען GPT-3.5 און GPT-4.

מען האט שוין פריער באהאנדלט אז די מאדעלן האבן מולטימאדאלע פעאיקייטן, דאס הייסט זיי קענען באארבעטן פארשידענע פארמען: טעקסט, בילדער און ווידעאס. זיי העלפן צו שאפן אינהאלט, צו מאכן א קיצור פון א טעקסט (text summarization), צו מאכן) sentiment analysisאפשאצן די שטימונג פון א טעקסט), צו ארויסנעמען דאטא פון א טעקסט און צו איבערזעצן פון איין שפראך אין אן אנדערע. די אלע ביישפילן האט מען געזען אין דער פריערדיגער לעקציע, למשל ווי אזוי מען זעצט איבער איין טעקסט אויף עטלעכע שפראכן. אין דער היינטיגער לעקציע וויל מען ווייזן ווי אזוי מען קען אויסנוצן די פעאיקייטן.

קיין LLM איז נישט "דער בעסטער" (8:39:20)

די פארשידענע מאדעלן זענען געבויט געווארן דורך פארשידענע פירמעס. יעדע פירמע שטעלט צו סערוויסעס פאר אירע קליענטן, און יעדע איינע זאגט אז זי איז די בעסטע. דער קורס באטאנט אז עס איז נישטא קיין פעסטע כלל וועלכער LLM איז דער בעסטער. דאס הענגט אפ אין דעם וואס מען דארף און ווי אזוי מען וויל אים ניצן.

די מאדעלן וועלן כסדר דערהיינטיגט. עס קען זיך מאכן אז באלד וועלן ארויסגעלאזט א נייע וועלכע וואס איז נאך בעסער ווי GPT-5, Gemini אדער Claude.

מען האט אויך דערמאנט א רשימה פון כלים. טייל זענען אומזיסט, טייל קאסטן געלט, און טייל קומען מיט א פראבע-אבאנעמענט פון איין חודש אדער 15 טעג. מען קען זיי אויספרובירן, אבער מען דארף פארשטיין די באדינגונגען פון יעדן איינעם.

וואס איז א פראמפט (8:40:21)

דער קורס דערמאנט די פריערדיגע אויפגאבע: מען האט מיט ChatGPT געשאפן א קורות-החיים (resume) און עס צוגעפאסט צו די פאדערונגען פון א געוויסע ארבעט.

דערנאך קומט די דעפיניציע פון א פראמפט. א פראמפט איז א טעקסט אין נאטירלעכער שפראך וואס העלפט אויספירן א געוויסע אויפגאבע. א גוטער פראמפט פירט צו א פאקוסירטן און ניצליכן רעזולטאט. ער איז די בריק צווישן דעם וואס דער באניצער וויל און דעם וואס דער AI פארשטייט.

א פראמפט דארף זיין געווי, ריכטיג און קלאר. דער קורס ברענגט א משל: עס איז ווי מען וואלט ערקלערט עפעס פאר א צווייי-יאר-אלט קינד. מען מוז עס זאגן קלאר, גוט אויסגעבויט און מיט א סדר. אן א ריכטיגן סדר און אן א פאסיגע לענג פארשטייט דער מאדעל נישט וואס מען וויל, און ער קען געבן אומקלארע רעזולטאטן.

ווי אזוי א פראמפט ארבעט, און פון וואנעט קומט די דאטא (8:41:22)

ווען מען שטעלט א גלייכע פראגע, גיט דער שפראך-מאדעל אן ענטפער אויפן סמך פון זיין טריינינג-דאטא. די פראגע ווערט געשטעלט צו א מאדעל וואס איז געטריינירט געווארן אויף א ריזיגע מאס דאטא. צווישן די מקורות:

- Common Crawl, דער דאטא-סעט C4
- Wikipedia
- ארויסגעצויגענע אינהאלט פון אינטערנעט-זייטלעך (scraping)
- נייעס-קוואלן און נאך

ווען די דאטא איז נישט דערהיינטיגט (8:42:24)

עס ווערט ערקלערט וואס עס פאסירט ווען מען פרעגט א LLM וועגן א פריש געשעעניש. דער ביישפיל: דער אינדישער פרעמיער-מיניסטער איז נאר וואס געפארן קיין א געוויסע לאנד. אויב מען פרעגט דעם LLM וועגן דעם, קען זיין אז ער וועט נישט קענען ענטפערן, ווייל די דאטא איז אים נישט אריינגעגעבן געווארן.

די דאטא גייט נאר ביז א געוויסן טאג. אויב היינט איז, למשל, דער 27טער יולי, קען זיין אז די דאטא איז דערהיינטיגט געווארן נאר ביזן 27טן יוני, א חודש צוריק. וואס עס איז פאסירט צווישן דעם 27טן יוני און דעם 27טן יולי איז נישט אריין אינעם מאדעל. דעריבער קען ער נישט דערציילן וועגן די געשעענישן פון יענער צייט, און ער קען געבן אן אומקלארן ענטפער.

מען דארף דאס געדענקען: ווען מען פרעגט עפעס וואס איז נישט געווען אין דער טריינינג-דאטא, אדער עפעס פון לעצטנס, וועט דער מאדעל אוודאי פרובירן צו ענטפערן, אבער דער ענטפער וועט נישט זיין ריכטיג.

האלוצינאציעס (8:43:58)

דער קורס ברענגט נאכאמאל דעם משל פון א צווייי-יאר-אלט קינד. אז מען פרעגט עס וועלכע לימודים עס האט געלערנט אין שולע, קען עס אנהייבן קוועלן און ענטפערן עפעס אנדערש אינגאנצן, למשל "איך האב נישט געגעסן" אדער "איך בין נישט געגאנגען אין שולע". דאס קינד גיט יא אן ענטפער, אבער דער ענטפער איז נישט ריכטיג.

אויף דעם זעלבן אופן, ווען א LLM ווייסט נישט דעם ענטפער, וועט ער פונדעסטוועגן שאפן עפעס. ער איז א געניראטיוור מאדעל, און ער מוז ארויסגעבן עפעס. דער באניצער איז דער ריכטער: מיט מענטשליכן שכל מוז ער משפטן צי דער ענטפער איז ריכטיג אדער פאלש.

דאס רופט מען האלוצינאציעס: ענטפערס וואס זענען נישט ריכטיג לויטן קאנטעקסט אדער לויט די פאקטן. "פאקטיש ריכטיג" מיינט אז דער ענטפער מוז אנטהאלטן ריכטיגע פאקטן. אויב מען פרעגט וואס פרעמיער-מיניסטער Modi האט באשמועסט ביי זיין לעצטן באזוך, ווייסט דער מאדעל נישט די פאקטן. ער קען אפשר געבן אן ענטפער וואס קלינגט ריכטיג לויטן באגריף, אבער לויט די פאקטן איז ער פאלש. דאס איז א האלוצינאציע.

די חלקים פון א גוטן פראמפט (8:45:29)

א גוטער פראמפט באשטייט פון פיר חלקים:

- אן אויפגאבע (task) – א געוויסע אקציע אדער פראצעס וואס מען וויל אויספירן.
- אנווייזונגען (instructions) – וואס דער LLM דארף טון און ביי וואס ער דארף זיך האלטן.
- קאנטעקסט (context) – הינטערגרונט-אינפארמאציע, וואס קען זיין אינפארמאציע וועגן דעם טעקסט, א בילד אדער עפעס אנדערש.
- אינפוט (input) – קען זיין א בילד, טעקסט אדער א דאטא-סעט.

דער אינפוט און דער קאנטעקסט זענען שטארק פארבונדן איינס מיטן אנדערן.

דער ביישפיל פון דער פריערדיגער לעקציע: די אויפגאבע איז געווען צו שאפן א קורצע דערציילונג. די אנווייזונג איז געווען אז זי זאל זיין וועגן א קוסמאכער'ס זוכעניש (wizard quest). דער קאנטעקסט איז געווען "אין א פארכישופטן וואלד". דער אינפוט איז געווען באשרייבונגען פון דער נאטור וואס מען האט צוגעשטעלט. אזוי אונטערשיידט מען צווישן די פיר חלקים פון א פראמפט.

דריי אקטיוויטעטן: איין LLM, פיל אויפגאבעס (8:47:02)

מען האט געטון דריי אקטיוויטעטן:

- איבערזעצונג אויף פארשידענע שפראכן – א טעקסט איבערגעזעצט אויף עטלעכע שפראכן.
- שאפן א LinkedIn-פראפיל – יעדער האט געשאפן אדער דערהיינטיגט זיין פראפיל.
- sentiment analysis.

די לעקציע פון דעם איז אז איין אייניציגער LLM קען דורכפירן פיל פארשידענע אויפגאבעס: קלאסיפיקאציע, שאפן טעקסט, און איבערזעצן פון איין שפראך אין אן אנדערע. דאס זענען אויפגאבעס וואס מען דארף טון אין טאג-טעגלעכן לעבן.

מולטימאדאלע פעאיקייטן (8:48:02)

דער קורס ערקלערט דעם באגריף "מולטימאדאל". עס ווערן אויסגערעכנט פינף "מאדאליטעטן" (וועגן פון באקומען אינפארמאציע) ביים מענטש: אנרירן, הערן, רעדן, טעקסט און בילד. פון די, קען א LLM באארבעטן הערן, רעדן, טעקסט און בילד.

מען קען אים געבן א טעקסט-דאקומענט, א בילד, אדער אויך רייד. רייד קען מען איבערמאכן אויף טעקסט (speech to text). דאס איז די כח פון די מאדעלן.

מען דערמאנט אויך א פריערדיגע אקטיוויטעט: מען האט אריינגעגעבן א בילד פון אמעריקע און געפרעגט וועלכע טוריסטישע ערטער מען קען באזוכן דארט און ווי אזוי צו פלאנירן א שטיקל צייט אין יענעם לאנד.

אקטיוויטעט: אנאליז פון קונה-באמערקונגען (8:49:06)

איצט גייט מען דורך די אקטיוויטעט פון אנאליזירן קונה-באמערקונגען (customer feedback). די באמערקונגען געפינען זיך אין א CSV-טעקע. דער ציל איז צו סארטירן זיי אין דריי קאטעגאריעס: פאזיטיוו, נעגאטיוו אדער קאנסטרוקטיוו.

דער הינטערגרונט: די באמערקונגען זענען געזאמלט געווארן נאך א גרויסן פארקויף-געשעניש. מען וויל דערקענען וואו מען האט מצליח געווען און וואו עס זענען דא געלעגנהייטן זיך צו פארבעסערן, כדי צו פארבעסערן די קומענדיגע פארקויף-סטראטעגיעס.

דער קורס פארגלייכט דאס מיט די פריערדיגע לעקציעס פון pandas און NumPy. דארט האט מען אפט געטון אזעלכע אקטיוויטעטן: מען האט געהאט פארקויף-דאטא פון קונים, און מען האט געוואלט פארגרעסערן דעם פארקויף און אויך פאראויסזאגן דעם קומענדיגן פארקויף. אנשטאט צו טון די אלע אפעראציעס מיט קאוד, קען מען פשוט אריינלייגן די CSV-טעקע אינעם LLM און אים בעטן: "ווייז מיר די רעזולטאטן אין א גראפיק, גיב מיר די שריט פון פריפראסעסינג" און אזוי ווייטער. ער פרובירט צו פארשטיין דעם קאנטעקסט פון דער CSV און ארויסגעבן רעזולטאטן.

די טעגלעכע גרענץ פון ChatGPT (8:50:40)

ביי איינעם פון די באטייליגטע האט ChatGPT אנגעזאגט אז די טעגלעכע גרענץ איז שוין דערגרייכט. דער קורס ערקלערט אז די אומזיסטע ווערסיע פון ChatGPT האט א טעגלעכע גרענץ, ווייל עס איז דא א גרענץ אויף די צאל tokens. נאך דעם באקומט מען א מעלדונג אז מען זאל פרובירן מארגן. עס קען אפילו פאסירן אז די גרענץ פונעם דעמאנסטראציע-חשבון אליין לויפט אויס אין מיטן דער לעקציע.

דער רעזולטאט פון דער אנאליז (8:51:40)

דער ענטפער פונעם מאדעל: דער דאטא-סעט אנטהאלט קונה-באמערקונגען מיט עלף קאלאמען, צווישן זיי review title, name, rating, date, category, comments און useful. אינעם comments-פעלד ווערן די באמערקונגען אויסגעטיילט אין פאזיטיוו, נעגאטיוו, קאנסטרוקטיוו און נייטראל. דער מאדעל האט אויך געוויזן די value counts, דאס הייסט וויפיל באמערקונגען עס זענען דא אין יעדער קאטעגאריע.

דערנאך האט דער מאדעל געמעלדעט אז ער קען יעצט נישט טון קיין מער פארגעשריטענע דאטא-אנאליז, און מען זאל פרובירן שפעטער. דער קורס ווארנט אז מען וועט אפט זען אזא מעלדונג. מען שיקט פארלאנגען צו זייערע סערווערס, און די סערווערס קענען זיין פארנומען. דאן זאגן זיי אז זיי קענען נישט טון קיין פארגעשריטענע אנאליז, ווייל דאס איז די אומזיסטע ווערסיע. דאס דארף מען פארשטיין.

טאָקענס, קאוד-געענעראַציע און ווערסיעס פון GPT (8:52:40)

פריי-ווערסיע, לימיטן און דער pie chart (8:52:40)

דער קורס ערקלערט אז די פירמע וואס שטייט הינטער ChatGPT פרובירט צו מארקעטירן איר סערוויס, און ווייל אז יעדער זאל באצאלן כדי עס צו קענען ניצן. דערפאר טרעפט זיך אמאל אז אין דער פרייער ווערסיע מאכט ChatGPT די אנאליז, און אמאל מאכט עס נישט; אמאל ווייזט עס אן אז מען האט שוין דערגרייכט דעם טעגליכן לימיט. עס ווערט דערמאנט אז מען האט גענוצט ChatGPT גאר א לאנגע צייט צוריק, ווען עס איז נאך געווען אין א beta ווערסיע.

אין דער לעקציע ווערט געוויזן א בייזפיל פון פריערדיגע איבונג: איינער פון די באטייליגטע האט באקומען דעם pie chart (אקיילעכדיגע גראפיק) גלייך פון ChatGPT, אבער ביי דער דעמאנסטראציע האט ChatGPT נישט געמאכט דעם pie chart, נאר אנשטאט דעם געגעבן א שריט-ביי-שריט Python סקריפט. מען קען קאפירן דעם סקריפט, אים לויפן אויפן אייגענעם קאמפיוטער, און אזוי זען דעם pie chart. נאך א באטייליגטער האט אויך באקומען א pie chart.

דער קורס גיט א טיפּ: אויב מען באקומט נישט די גראפיק, קען מען בעטן ChatGPT בפירוש, למשל: "קענסט מיר געבן א Python קאוד-שטיקל צו לויפן לאקאל, וואס גענערירט א pie chart פאר די קאטעגאריעס פון די קונים-פידבעק? קענסט גענערירן דעם קאוד שריט ביי שריט?" וועט מען בעט אזוי דעם קאוד, וועט עס גענערירן דעם קאוד. דערנאך דארף מען זיכער מאכן אז די נויטיגע Python פעקלעך (packages) זענען אינסטאלירט, און לויפן דעם קאוד און זען אליין.

קאוד-געניראַציע – שטארק, אבער מיט זהירות (8:54:14)

עס ווערט ערקלערט אז דאס איז א בייזפיל פון דעם וואס מען האט שוין געריידט אין דעם טעאָרעטישן טייל: א גרויסער שפראך-מאדעל קען נישט נאר מאכן) sentiment analysisאנאליז פון געפילן אין טעקסט), סאמעריזאציע (קיצור) און טעקסט-געניראַציע, און נישט נאר ארבעטן מיט מולטימאדאַליטעט, נאר אויך קאוד-פארענדיגונג (code completion) און קאוד-געניראַציע.

דער קורס באטאָנט אבער: דאס איז זייער שטארק, אבער מען דארף עס ניצן מיט זהירות און אויך ניצן דעם אייגענעם מוח. וואס די מאדעלן זאגן איז אפט ריכטיג, אבער דער באניצער איז דער ריכטער. מען טאר נישט אננעמען בלינד אז ChatGPT און די LLM מאדעלן זענען שטענדיג געריכט; מען דארף אמאל פארגלייכן און באשטעטיגן די דאטא. דערפאר דארף מען א מענטשליכן אפשאצער (human evaluator), א מומחה, כדי אפצושאצן די ChatGPT און גרויסע שפראך-מאדעלן.

עס ווערט דערציילט פון א נייעס: איינער האט לאנגע צייט געוצט ChatGPT אדער אן LLM מאדעל אין זיינע קאוד-באזעס. איינער האט געהייסן ChatGPT לויפן א געוויסן סקריפט, און דורך א טעות האט עס פרובירט אויסמעקן אלע איינסן (entries), און דערנאך ארויסגעגעבן א מעלדונג אז "איך האב אויסגעמעקט די ארבעטער". דאס קען פאסירן, ווייל עס איז א געניראַטיוווער מאדעל וואס טייטשט אויס די פראגע. דערפאר דארף דער פראמפט זיין קלאר ווי קרישטאל און גענוי, כדי עס זאל נישט פאלש פארשטיין; וועז די אויסטייטשונג איז פאלש, קען עס ברענגען אנטוישנדיגע רעזולטאטן, ווי דאס אויסמעקן אלע איינסן אין דער דאטא-באזע, וואס קען האבן א ריזיגע השפעה.

די עצה: אויב מען ניצט אזא זאך, זאל מען עס קודם פרובירן, און ערשט דערנאך ניצן אין דער production סביבה. נישט ניצן בלינד. לויט ווי דער קורס פארשטייט, האבן יענע מענטשן געוצט אזעלכע געצייג וואס ווערן איינגעבויט אין יעדן) IDEדי ארבעט-סביבה פאר שרייבן קאוד); וועז זיי האבן געשריבן קאוד אין IDE, האבן זיי געבעטן דעם LLM טון די ארבעט פאר זיי, און אין דעם פראצעס איז דאס פאסירט מיט די דאטא-באזע איינסן.

פארגלייך פון GPT וועריסעס; וואס זענען טאָקענס (8:56:47)

דער נעקסטער טייל איז "אויספארשן שטארקע GPTs ": פארגלייכן די חילוקים צווישן פארשידענע סארטן GPT, ווי GPT 3.5, GPT 4 און turbo. ביי די נייערע וועריסעס איז די) context lengthוויפיל טעקסט דער מאדעל קען האלטן אויף איינמאל) און די צאל טאָקענס פארגרעסערט געווארן.

דער קורס חזרט איבער וואס טאָקענס זענען, וואס איז ערקלערט געווארן אין דער פריערדיגער לעקציע. באטייליגטע האבן געענטפערט אז די ווערטער און ספעיסעס ווערן אנגעצייכנט אין פארשידענע פארבן, און אז עס האט צו טון מיט ווערטער, צאל אותיות און ספעיסעס. עס ווערט ערקלערט אז וואס מען האט פריער געוויזן מיט די פארבן איז נאר א פארשטעלונג. דער אמת'ער טאָקען איז אז א וואָרט ווערט פארוואנדלט אין א נומעריש פארשטעלונג.

א בייזפיל: דער טעקסט "chat GPT-3.5". למשל, "chat" באקומט א נומער ווי 15, "GPT" ווערט 45, דער שטריכל (dash) ווערט 20, און "3.5" ווערט 113. אזעלכע נומערישע פארשטעלונגען רופט מען טאָקענס. ווען מען גיט ארויף אזא טעקסט, גייט עס אריין ווי א מערך (array) פון נומערן, למשל 56, 15, 45, 20 און אזוי ווייטער, וואס שטייען פאר די טיילן פון טעקסט.

א צווייטער בייזפיל: דער זאץ "I want to go to college " ווערט צעטיילט אין זעקס טאָקענס, יעדער מיט א token ID. למשל 40 שטייט פאר "I", 1682 פאר "want", 316 פאר "to", 810 פאר "go", דערנאך ווידער "to", און דערנאך "college".

ווי ווערן טאָקענס צעטיילט – Byte Pair Encoding (8:59:18)

אויף א פראגע פון א באטייליגטער צי עס איז דא א לאגיק ווי די נומערן ווערן צוגעטיילט, ענטפערט דער קורס: יא. זיי ווערן נישט סתם צוגעטיילט, נאר דורך אן אלגאריתם, וואס הייסט byte pair encoding (BPE). דאס איז דער אלגאריתם וואס ווערט געניצט אין GPT-3; ווען GPT-4 און 4 mini איז נישט באקאנט, ווייל מיט דער צייט האבן זיך אנטוויקלט פארשידענע ווערסיעס פון דעם אלגאריתם.

מען זעט א דעמא: מען שרייבט א זאץ און זעט אז "I" איז איין טאָקען, "m" איז איין טאָקען, "going" איז איין טאָקען, "to" איז איין טאָקען, אבער א וואָרט ווי "calling" ווערט צעטיילט באזונדער אין "call" און "g". יעדע פארב שטעלט פאר א באזונדערן טאָקען. אזוי אויך "LLM" ווערט צעבראכן אין טיילן ווי "L" און "LM". די לאגיק איז אז זיי צעטיילן ווערטער אין שטיקער. א בייזפיל: "come" און "coming" – ס'קען זיין א שטראטעגיע וואו "coming" ווערט צעבראכן אין צוויי טאָקענס (דער שורש און "ing"), בשעת "come" איז איין טאָקען. עס זענען דא פארשידענע שטראטעגיעס און אלגאריטמען פאר טאָקעניזאציע.

א כלל אצבע: 100 טאָקענס זענען בערך 75 ווערטער. ווען מען שרייבט א דאקומענט פון 75 ווערטער, ווערט עס האבן נאענט צו 100 טאָקענס. מען קען טענה'ן אז 75 ווערטער זאלן זיין 75 טאָקענס, איין טאָקען פער וואָרט, אבער דאס איז נישט אזוי. אין דעם בייזפיל אויפן עקראן זענען דא אכט ווערטער, אבער דער כלי האט אידענטיפיצירט 11 טאָקענס. דאס איז דער OpenAI tokenizer, וואס רעכנט אויס וויפיל טאָקענס א טעקסט באטרעפט, און די אמת'ע פארשטעלונג איז נישט די פארבן נאר די token IDs.

דער גאנצער פראצעס: טעקסט → טאָקענס → LLM → טעקסט (9:02:27)

דער קורס שרייבט אויס דעם גאנצן פראצעס:

- מען האט א אריינגאנג-טעקסט (input text).
- מען פארוואנדלט אים אין טאָקענס, דאס הייסט א מערך פון נומערן.
- מען גיט די טאָקענס ארויף צום LLM.
- דער LLM גענערירט טאָקענס אלס ענטפער.
- די טאָקענס ווערן צוריק דעקאדירט אין טעקסט, וואס דער באניצער זעט אויפן עקראן.

פראגעס: טעגליכע לימיטן און טאָקענס אין פארשידענע ווערסיעס (9:03:27)

פראגע: וויפיל טאָקענס קען מען ניצן טעגליך מיט דער פרייער ChatGPT? דער ענטפער: מען דארף עס נאכקוקן אין ChatGPT אליין; ס'קען זיין א טעגליכער לימיט, אבער ער טוישט זיך כסדר. מען ווייסט עס נישט גענוי, ווייל די ווערסיע וואס מען האט גענוצט איז געווען די beta ווערסיע, ווען מען האט ארויסגעלאזט פאר עטליכע הונדערט אדער 2,000 קונים; מען האט אריינגעגעבן אן אימעיל און באקומען א לינק.

פראגע: וועט די צאל טאָקענס פאר דעם זעלבן טעקסט זיין אנדערש אין פארשידענע וועלטן? ענטפער: יא, עס הענגט אפ אינעם אלגאריטם. למשל, דער טעקסט וואס האט 11 טאָקענס אין איין וועלט, קען האבן אן אנדערע צאל אין 3.5, ווייל 3.5 קען ניצן אן אנדערן אלגאריטם אדער שטראטעגיע; ס'קען אבער אויך זיין אז עס וועט נישט זיין אנדערש. פאר GPT-3, ווייט ווי באקאנט, ניצט מען byte pair encoding. דער נאמען קומט דערפון וואס יעדעס וואָרט וועלט פארוואנדלט אין bytes און דערנאך באנוצט. עס זענען אויך דא אנדערע אלגאריטמען:) character-wise encodingלויט אותיות) און) sentence-wise encodingלויט זאצן). עס הענגט אפ וועלכן אלגאריטם GPT-4o ניצט. GPT-3 איז שוין) legacyאלט און מער נישט אין באנוץ).

שנעלקייט, לאנגע קאנטעקסט, און פארשידענע ענטפערס (9:05:32)

דער קורס חזרט איבער: GPT-3 איז שוין legacy, און היינט האט מען 3.5, 4, און 4o אדער 3.5 turbo. גיט שנעלערע ענטפערס; 4 איז פאמעלעכער, אבער האנדלט לענגערע קאנטעקסט און גיט מער צוזאמענהענגדיגע סאמעריס. ווייט ווי באקאנט, קען GPT-4 האלטן אין זיין קאנטעקסט א בוך פון 300 זייטן, למשל א PDF פון 300 זייטן.

לויט די וועלט קען מען באקומען פארשידענע ענטפערס. אויך אויב איינער פרעגט א פראגע און א צווייטער פרעגט די זעלבע פראגע, מוז דער ענטפער נישט זיין דער זעלבער. דאס האט מען שוין געזען: דער pie chart איז נישט געווארן ביי דער דעמאנסטראציע, אבער יא ביי איינעם פון די באטייליגטע. די רעזולטאטן קענען זיך טוישן לויט ווי מען שרייבט דעם פראמפט. אבער ביי פראגעס מיט פעסטע ענטפערס, ווי "וואס איז די הויפטשטאט פון פראנקרייך?" אדער "וועט איז דער פרעמיער מיניסטער פון אינדיע?", וועלן די רעזולטאטן נישט טוישן – זיי זענען קאנסיסטענט. ביי טעקסט-סאמעריזאציע אבער קענען די רעזולטאטן זיך טוישן, לויט ווי דער מאדעל ארבעט און לויט דער וועלט.

דער קורס ענדיגט אז דא האט מען זיך אפגעשטעלט אין דער פריערדיגער לעקציע, און פון יעצט וועט מען זיך קאנצענטרירן אויף שרייבן פראמפטס.

שרייב-געצייג און דיזיין מיט Canva AI (9:07:34)

Write For Me: א פערזענליכער שרייב-אסיסטענט (9:07:34)

דער קורס הייבט אן מיט א געצייג וואס הייסט Write For Me. דאס איז א געצייג פון ChatGPT וואס ארבעט ווי א פערזענליכער שרייב-אסיסטענט. מען קען עס ניצן פאר שאפן אינהאלט, פאר הילף ביים שרייבן און פאר רעדאקטירן.

עס ווערט ערקלערט מיט א ביישפיל: מען וויל שרייבן אן אימעיל. מען האט שוין דעם קאנטעקסט, נאר מען וויל אז דער בריוו זאל זיין געווי פארמולירט און אויסגעדריקט אין א באשיידענעם טאן. למשל, מען וויל זאגן פארן מענעדזשער אז מען בעט א פרייע צייט ווייל מען פילט זיך נישט גוט. אדער מען האט געמאכט א פרעזענטאציע פון א פראיעקט, און מען וויל שרייבן וואס זענען געווען די רעזולטאטן און וואס מען האט געפונען. אין אזעלכע פעלער איז דאס א נוצליך געצייג, ווייל עס שרייבט דעם אימעיל און פארמולירט אים שיין.

א פראגע: ווי אזוי ווערן טאָקענס גערעכנט (9:08:30)

דא איז געקומען א פראגע: אויב מען קאפירט אינהאלט, למשל מען קאפירט א רעזומע און מען לייגט עס אריין אין ChatGPT, רעכנט מען דאס אויך אין די טאָקענס?

דער קורס ערקלערט עס מיט א ביישפיל. די רעזומע וואס מען לייגט אריין איז א טייל פונעם פראמפט, דאס הייסט פונעם אריינגאנג. לאמיר אננעמען אז דער אריינגאנג האט ארום 200 טאָקענס. דער ענטפער וואס דער מאדעל גיט צוריק האט, לאמיר זאגן, 150 טאָקענס. מען לייגט זיי צוזאמען, און דאס איז די צאל טאָקענס וואס מען האט אויסגענוצט: 350 טאָקענס זענען "אוועק".

דער מאמענט וואס מען שרייבט דעם אריינגאנג, ווערט אויסגערעכנט וויפיל מען פרעגט. דער ענטפער וואס דער מאדעל שאפט ווערט אויך אויסגערעכנט. אויב מען האט אריינגעלייגט א רעזומע און דערצו געשריבן א פראמפט, ווערן דריי זאכן גערעכנט אויף די טאָקענס:

- די רעזומע
- דער פראמפט
- דער ענטפער

דער Writer-סייט און זיינע איינשטעלונגען (9:10:41)

דערנאך ווערט געוויזן דער ChatGPT-שרייבער. מען גייט צום סייט chatgptwriter.ai/app. מען זעט דארט אן אינטערפייס וואס זעט אויס גוט. ווי א ביישפיל שטייט דארט א פארשלאג: "שרייב אן אימעיל וואס בעט צופרידענע קליענטן אז זיי זאלן רעקאמענדירן אדער שרייבן אן עדות-בריוו (testimonial)."

עס זענען דא עטלעכע איינשטעלונגען:

- מאדעלן: מען קען אויסוועלן צווישן פארשידענע מאדעלן, למשל ChatGPT mini, Gemini, DeepSeek און DeepSeek reasoning. מען קען פרובירן מיט אנדערע מאדעלן אויך. ביי יעדן מאדעל ווערט אנגעוויזן א "ראנג" פאר אינטעליגענץ און א ראנג פאר גיכקייט, למשל ראנג איינס אדער ראנג צוויי. טייל פון זיי זענען בצאלט.
- שרייב-סטיל: פראפעסיאנעל, לייכט (casual), גלייך צום פונקט, איבערצייגנד אדער פריינטלעך.
- לענג: קורץ, מיטל אדער לאנג.

א פראקטישע איבונג: א בלאג-ארטיקל פאר א וואסער-פלאש (9:11:43)

מען טוט א פראקטישע איבונג. מען בעט אז דער מאדעל זאל שרייבן א בלאג-ארטיקל פאר א נייעם פראדוקט: אן אקא-פריינטלעכע וואסער-פלאש וואס מען קען ניצן ווידער און ווידער. דער פראמפט גיט דריי אנווייזונגען:

- דער ארטיקל זאל פארשטעלן דעם פראדוקט און זיינע ספעציעלע אייגנשאפטן.
- ער זאל באשפרעכן פארוואס עס איז וויכטיג צו פארמינערן פלאסטיק וואס ווערט געוצט נאר איין מאל אין טעגליכן לעבן.
- ער זאל ארויסברענגען די געזונטהייט- און סביבה-מעלות פון אויסקלייבן די בארימטע (sustainable) פלאש.

דער סטיל דארף זיין אינפארמאטיוו, ציענדיג און איבערצייגנד. דער טאן דארף רעדן צו מענטשן וואס היטן זייער געזונט און וואס זענען אויפמערקזאם אויף דער סביבה. מען קען אויסוועלן ChatGPT, Gemini אדער DeepSeek, און מען קען פרובירן מיט יעדן.

עמעצער האט געפרעגט ווי אזוי מען עפנט דעם Writer-אפליקאציע. דער ענטפער איז אז מען דריקט פשוט אויפן לינק. עס איז אויך דא אן אפליקאציע פון ChatGPT Writer וואס מען קען אראפנעמען און אינסטאלירן. וועו מען שרייבט אן אימעיל, באווייזט זיך דאס געצייג גלייך דארט.

דער אונטערשייד צווישן אלגעמיינעם ChatGPT און ספעציפישע GPTs (9:13:47)

דער קורס ערקלערט דעם אונטערשייד. דער ChatGPT וואס מען האט פריער געוענט איז דער געוועלכער ChatGPT. ער קען אויספירן אלגעמיינע אויפגאבעס. דער Writer, אנדערערזייטס, איז ספעציעל געבויט פאר שרייבן, און ער איז א מער פארבעסערטע ווערסיע פאר דעם צוועק. דאס זעלבע וועט מען זען ביי Canva AI, וואס ווערט געוענט פאר דיזיין. פאר ספעציפישע אויפגאבעס האט מען געבויט ספעציפישע GPTs אויף ChatGPT, וואו מען קען שטעלן פראגעס, און דאס איז זייער נוצליך.

פריער האט מען געוענט Grammarly כדי צו פארריכטן טעותים אינעם שרייבן. היינט קען מען בעטן דעם מאדעל אז ער זאל שאפן אן אימעיל מיט א פריינטלעכן טאן, שיין פארמאטירט און מיט ריכטיגן גראמאטיק.

מען זעט אויפן עקראן אז פון 15 אומזיסטע ענטפערס זענען שוין צוויי אויסגענוצט. נאכדעם בעט דאס געצייג אז מען זאל זיך אפגרעידן. די מסקנה פונעם קורס: עס איז נישטא קיין אומזיסטע מאלצייט, און מען דארף זיין פארזיכטיג וועו מען ניצט די GPTs.

צוזאמגעפאסט: Write For Me איז א פערזענליכער שרייב-אסיסטענט און "קאוטש". מען קען אים ניצן וועו מען שרייבט א טאגבוך, וועו מען וויל וויסן צי דער טעקסט איז גראמאטיש ריכטיג, וועו מען שרייבט אן עסיי, א צוזאמפאסונג אדער א געדאנק.

Canva AI: דיזיין פון א באשרייבונג (9:16:23)

דאס נעקסטע געצייג איז Canva AI. עס העלפט פארגרעסערן די עפעקטיווקייט ביי אויפגאבעס און ארבעט-פראצעסן. Canva מאכט דיזיין גרינג, ווייל עס שאפט דיזיינס פון א באשרייבונג. מען האט א געדאנק אין קאפ, מען שרייבט אים אראפ ווי א באשרייבונג, און Canva פארוואנדלט די באשרייבונג אין א שעפערישן דיזיין. מען געפינט דאס געצייג אויפן אינטערנעט דורך זוכן דעם לינק.

דער פראמפט אין דער דעמא איז: שאף א בילד פאר א פאסט אין די סאציאלע מעדיע, כדי אנצוזאגן אז מען לאנצירט "Sunrise Brew". דער פראמפט באשרייבט וואס דאס בילד דארף ענטהאלטן:

- עס דארף ווייזן די ווארעמקייט פון דער פרימארגן-זון און דעם רייכן טעם פון קאראמעל און שאקאלאד.
- עס דארף ניצן עלעמענטן ווי א זון-אויפגאנג אדער פרימארגן-ליכט, און א דאמפנדיגע טעפל קאווע, כדי צו אנדייטן א פרישן אנהייב פונעם טאג.
- דער דיזיין דארף זיין אויפנעמעריש און דערפרישנדיג.
- עס דארף ענטהאלטן דעם טעקסט "Start your day with flavor and – Introducing Sunrise Brew adventure" אין א קלארן פאנט. מען קען אויך אנגעבן דעם פאנט, די ווייטקייט צווישן די אותיות, די גרייס און אזוי ווייטער.
- דער סטיל דארף זיין מאדערן און ריין, און געצילט פאר יונגע פראפעסיאנאלן און קאווע-ליבהאבער.

דער קורס באטאנט פארוואס ער גיט דווקא אזא פראמפט: מען זאל זען ווי אזוי ער איז פארמולירט, מיט געוויע און קלארע אנווייזונגען, וואס זאגן דעם מאדעל פונקט וואס ער דארף טון און וועלכע עלעמענטן ער דארף ניצן.

ווי אזוי דער Canva-פלאגין ארבעט (9:19:00)

וועו מען שיקט אוועק דעם פראמפט, לאדט ChatGPT אריין אן עקסטערנעם פלאגין, דעם Canva-פלאגין. ער גייט אריין אינעם Canva-סייט, ער "רעדט" מיט אים, און ער שטופט די פראגע אריבער צו דער אנדערער פלאטפארמע. נאכדעם נעמט ער צוריק דעם ענטפער. אויפן עקראן שטייט "Talk to ChatGPT plugin canva.com". דאס הייסט, מען ניצט עקסטערנע געצייג: מען פירט אריבער דעם אריינגאנג צו אן אנדער געצייג, דאס געצייג פראצעסירט אים, און דער רעזולטאט ווערט צוריק געוויזן אין ChatGPT.

מען באקומט פיר אפציעס פון דיזיינס. לויטן קורס זעען זיי אויס גוט. וועו מען וואלט געווואלט עפענען א קאווע-פירמע, קען מען ניצן איינע פון זיי, למשל די וואס האט אויך טעקסט. דעם טעקסט קען מען רעדאקטירן.

פראגעס פונעם עולם (9:20:37)

עמעצער האט געפרעגט צי דאס איז אויך פאראן אין טעלעפאן-אפליקאציעס. דער ענטפער איז אז ChatGPT איז פאראן אויפן טעלעפאן, אבער דאס האט מען נישט געפרואווט. מען זאל עס אליין נאכקוקן.

אנדערע האבן געזאגט אז ביי זיי קומען ארויס אנדערע אפציעס. עס ווערט ערקלערט אז, ווי שוין געזאגט, די רעזולטאטן וואס ווערן געשאפן קענען זיין אנדערש פאר פארשידענע מענטשן. דאס הענגט אויך אפ אין וועלכע ווערסיע פון ChatGPT מען ניצט.

א צווייטע פראגע איז געווען וועלכעס געצייג פאסט פאר שאפן א פלאן פון א דירה (floor plan) און פאר אינערליכן דיזיין. דער ענטפער: Canva קען פאסן, אבער מען האט נישט אויסגעפארשט וועלכעס געצייג איז דאס בעסטע פאר דעם צוועק. מען וועט עס אפשר נאכזוכן און זאגן אין דער קומענדיגער לעקציע. אין דער פרעזענטאציע (PPT) וואס מען האט צעטיילט זענען דא פילע פראדוקטיוויטעט-געצייג פאר דיזיין, למשל Looka (לאגא-DALL ,E·AI און Stable Diffusion. אפשר ארבעט עפעס א סטארטאפ ספעציעל אויף דיזיין פון הויז-פלענער און אינערליכן דיזיין. מען זאל אויספארשן די געצייג און זען וואס פאסט בעסטן פאר די אייגענע באדערפענישן.

צום סוף ווערט באמערקט אז Canva ווערט בריט געוצט פאר שאפן מוסטערן (templates) און פאר שאפן וועבסייטס. מען קען אויך שטעלן די זעלבע פראגע צו ChatGPT גלייך.

ספעציאליזירטע GPT געצייג: דיזיין, פארשונג און לערנען (9:23:14)

פארוואס א ספעציאליזירט געצייג? און דער Designer GPT (9:23:14)

דער קורס ערקלערט פארוואס מען באניצט זיך מיט ספעציאליזירטע געצייג אנשטאט צו פרעגן אלעס בלויז ביים געווענליכן ChatGPT. אזא געצייג פירט די פראגע וויטער צו א געוויסן פעלד, ווי למשל דיזיין. מען קען פרעגן דיזיין-פראגעס אויך ביי ChatGPT, און מען וועט באקומען ענטפערס, אבער די ענטפערס וועלן זיין אנדערש.

עס ווערט געגעבן א משל: אויב מען פרעגט א דיזיינער א פראגע וועגן קאוד, וועט ער מעגליך נישט ענטפערן ריכטיג. פרעגט מען אים אבער א דיזיין-פראגע, איז זייער מעגליך אז ער וועט געבן דעם ריכטיגן ענטפער. אזוי איז עס מיט די געצייג: Canva איז נאר פאר דיזיין, און וועט עס דארף עפעס דיזיינען, פאר דעם איז דאס דאס בעסטע געצייג. פאר א זשורנאל איז ChatGPT דאס בעסטע. מיט "Write For Me" קען מען א געשריבענעם טעקסט אויסברייטערן און אנטוויקלען, כדי מען זאל אים קענען ניצן.

דערנאך ווערט פארגעשטעלט א געצייג פאר קאוד: אויטאמאטיש שאפן קאוד, איבערקוקן קאוד (code review), און אפילו לערנען פראגראמירן. דער קורס זאגט אז Designer GPT איז א ספעציאליזירטער AI, וואס שאפט און הייסט (hosting) גלייך שיינע HTML וועב-בלעטער, צוגעפאסט צו וואס דער באניצער בעט. למשל, מען האט אין זינען א וועבזייטל און מען וויל אז דער מאדעל זאל שרייבן דעם קאוד. דעם קאוד קען מען דערנאך ניצן צו בויען שיינע וועבזייטלעך וואס טוען וואס מען דארף.

א דעמא: א לאנדינג-בלאט פאר אן אנליין געשעפט (9:25:17)

אין דער לעקציע ווערט געוויזן א פראקטישער פראמפט. מען בעט פון Designer GPT א דיזיין-קאנצעפט פאר א וועבזייטל אדער א "לאנדינג-בלאט" פון א פראדוקט אין אן אנליין געשעפט. דער פראמפט פאדערט אז דער דיזיין זאל אנטהאלטן:

- א קאפ (header) מיטן לאגא פונעם געשעפט;
- א נאוויגאציע-ליניע (navigation bar);
- א "hero section" וואס שטעלט ארויס דעם הויפט-פראדוקט;
- אפטיילן פאר די פרטים פונעם פראדוקט און פאר עדות פון קונים (testimonials);
- א פוס (footer) אונטן.

דערצו פאדערט דער פראמפט אז דער דיזיין זאל זיין שיין פארן אויג, מיט א באלאנסירטער פארבן-סכעמע און א קלארער טיפאגראפיע.

נאכן אריינשיקן, ווערט געוויזן אז דאס געצייג גיט נישט גלייך דעם קאוד. ערשט פרעגט עס וועלכע בילדער מען וויל ניצן פארן hero section און פאר די פראדוקט-בילדער. מען ענטפערט "E·DALL", אז די בילדער זאלן געשאפן ווערן מיט E·DALL. די וואס זענען מיטגעגאנגען אין דער לעקציע האבן באקומען זייער שיינע און רעאליסטישע דיזיינס. אויף דער סיסטעם פון דער לעקציע איז דאס שאפן בילדער אבער נישט געלונגען, ווייל מען האט זיך שוין פריער אסאך באנוצט מיטן געצייג. דאס געצייג האט דערנאך פארגעשלאגן צו גיין ווייטער מיט דעם HTML-אויסשטעל (layout) און צוגעבן א בילד, און מען האט געענטפערט: יא, גיי ווייטער און גיב דעם HTML-layout.

דער רעזולטאט: דאס געצייג האט געגעבן דעם גאנצן קאוד, מיטן לאגא, דעם hero section, די ליסטע פון פראדוקטן, דעם הויפט-אינהאלט (main content), די פראדוקט-אייגנשאפטן (product features) און דעם footer. דער קורס דערמאנט אויך אז Replit איז נאך א גוט געצייג פון די LLM-געצייג וואס זענען פריער געווזן געווארן, און מען קען עס אויספרובירן.

דער "placeholder", און וואס מען געווינט דערפון (9:27:50)

עס ווערט ערקלערט אז ביז דאס E·DALL-בילד איז גרייט, שטעלט דער קאוד א "placeholder", א צייטווייליגן פלאץ פאר דעם בילד. אין קאוד זעט מען דעם מקור פונעם בילד (image source) מיט א לינק, און דארט קען מען אריינלייגן דאס אייגענע בילד און זען ווי עס קוקט אויס. מען קען אויך טוישן דעם נאמען פונעם געשעפט און די פרטים פון די פראדוקטן, פשוט אריינשרייבן וואס מען וויל.

דער קורס באטאנט אז דאס שפארט א סך צייט און מי ביים בויען דעם דיזיין, און דעם קאוד גיט עס אויך. מען זעט אז מיט איין פראמפט, וועלן צען מענטשן שרייבן אים אין דער זעלבער צייט, באקומט מען צען פארשידענע

דיזיינס. די בילדער זענען זייער איינדרוקספול און די דיזיינס זענען אויס רעאליסטיש. מיט אזא פשוטן פראמפט קען מען באקומען רעאליסטישע דיזיינס און דערויף בויען דאס וועבזייטל.

Consensus: א פארשונגס-געהילף פאר וויסנשאפטלעכע ארבעטן (9:29:20)

דאס נעקסטע געצייג איז פאר פארשונג און אנאליז: Consensus. דער קורס באשרייבט עס ווי א וויסנשאפטלעכן פארשונגס-געהילף, וואס ניצט AI צו צוזאמענשטעלן (synthesize) תוכן פון אקאדעמישע ארבעטן. אז מען וויל עפעס אויספארשן, צי מען וויל לייזן א געוויסע פראבלעם און מען דארף דערפאר פארשונג, גיט Consensus קורצע, קלארע סיכומים און ענטפערס וואס זענען געבויט אויף באווייזן (evidence-based) צו די פראגעס.

דער סצענאר: א פארשער מאכט א "literature survey", א דורכקוק פון דער ליטעראטור, וועגן דער השפעה פון גרויסע שפראך-מאדעלן (LLMs) אויף אויפגאבן פון פארשטיין נאטירלעכע שפראך (NLU). מען וויל צונויפקלייבן און אנאליזירן נייע שטודיעס וועגן דעם פארשריט פון LLMs, בפרט אין פעלדער ווי ענטפערן פראגעס (question answering), קירצן טעקסטן (text summarization) און סענטימענט-אנאליז (sentiment analysis). דער פארשער וויל אויסגעפינען די הויפט-טרענדס, די שווערigקייטן און די מעגליכע ריכטונגען פאר דער צוקונפט. ער וויל אויך ארויסברענגען וואספארא ראלע fine-tuning און transfer learning שפילן אין פארבעסערן ווי די LLMs ארבעטן אויף פארשידענע NLU-אויפגאבן.

דער פראמפט, געבויט אויף דעם הינטערגרונט: מאך א literature survey וועגן דער השפעה פון גרויסע שפראך-מאדעלן, מיט א פאקוס אויף נייע שטודיעס וועגן question answering, text summarization אדער sentiment analysis. אנאליזיר די הויפט-טרענדס און שווערigקייטן, און גיב א סיכום פון די געפינסן, מיט די מעגליכע ריכטונגען פאר דער צוקונפט אין דעם פעלד.

די רעזולטאטן פון Consensus (9:31:55)

מען זעט אז דאס געצייג האט געגעבן א קלארן סיכום, צעטיילט לויט די אויפגאבן: sentiment analysis, question answering און text summarization. עס האט אויך געגעבן לינקס צו די ארבעטן (papers) אליין, כדי מען זאל זיי קענען קוקן און נאכפרופן. דער קורס באטאנט אז ווען מען ארבעט אויף א פארשונג, איז די ערשטע און וויכטיגסטע זאך צו זאמלען אינפארמאציע: וואס ווערט שוין אנטוויקלט און וואס פאר א פארשונג גייט פאר אין דעם פעלד. מען מוז מאכן א "מארק-אנאליז" פון וועלכע ארבעטן זענען שוין ארויס, און לויט דעם אויסגעפינען אין וועלכער ריכטונג מען קען עפעס צולייגן.

דער סיכום האט אנטהאלטן:

- פארשריט אין question answering, מיט די שייכותדיגע ארבעטן;
- ווי גוט די מאדעלן ארבעטן אין text summarization, מיט ארבעטן;
- "multilingual" און "low-resource adaptation", צופאסן מאדעלן צו פיל שפראכן און צו שפראכן מיט וויניג דאטא, מיט נייע ארבעטן פון 2023 און 2025;
- שווערigקייטן און לוקן: שוואכערע רעזולטאטן אין low-resource מצבים, "hallucinations" (ווען דער מאדעל דערפינדט זאכן), און פראבלעמען מיט פעסטקייט (robustness);
- צוקונפט-ריכטונגען: אראפברענגען פארטייאישקייט (bias mitigation) און ערקלערבארקייט (explainability).

Universal Primer: לערנען אלעס וועגן יעדע זאך (9:33:25)

דאס נעקסטע געצייג איז Universal Primer, וואס באשרייבט זיך אלס "דער שנעלסטער וועג צו לערנען אלעס וועגן יעדע זאך". מען קען עס אויך אליין אויספרובירן מיט אן אייגענעם סצענאר.

דער סצענאר אין דער לעקציע איז קלאר מאכן דעם באגריף פון כעמישן גלייכגעוויכט (chemical equilibrium), אנגעהויבן מיט הויכשול-שילער. אין דעם סצענאר איז Alex א בעל-מלאכה מיט אן אינזשעניר-הינטערגרונט, וואס גרייט צו א וויכטיגע פרעזענטאציע פון א פראיעקט, און דארט דארף ער פארשטיין די כללים פון כעמישן גלייכגעוויכט. טראץ זיין עקספיריענס, האט ער שוועריגקייטן צו כאפן ווי ענדערונגען אין דרוק, טעמפעראטור און קאנצענטראציע ווירקן אויף כעמישן גלייכגעוויכט לויטן Le Chatelier's principle. ער זוכט א פשוטע ערקלערונג מיט ביישפילן פון פאקטישן לעבן, כדי צו פארטיפן זיין פארשטאנד און עס קענען ניצן אין זיין פראיעקט.

דער פראמפט: גיב א פראקטישע ערקלערונג פון Le Chatelier's principle און זיין שייכות צו כעמישן גלייכגעוויכט, צוגעפאסט פאר בעלי-מלאכה, און באשרייב די ווירקונג פון דרוק, טעמפעראטור און קאנצענטראציע אויף כעמישן גלייכגעוויכט. מען לייגט אריין דעם פראמפט גלייך, און דאס געצייג גיט א קורצע איבערזיכט פון אויבן, מיט א משל, ווי מען ניצט דעם פרינציפ, מיט א טעכנישן צוגאנג פאר אן אינדוסטריעלן אדער אינזשעניר-הינטערגרונט. דער קורס מוטיגט אז מען קען שרייבן אייגענע פראמפטס, יעדעס מאל וואס מען וויל עפעס לערנען אדער מען איז נייגעריג ווי עפעס ארבעט.

איבערחזרן און וואַרענונג (9:36:01)

עס ווערט איבערגעחזרט אז מען האט זיך באנוצט מיט פיר-פינף פארשידענע געצייג: ChatGPT Write For Me, Canva, Designer GPT, Consensus און Universal Primer. מען קען יעדע זאך פרעגן אויך ביים געווענליכן ChatGPT, אבער די געצייג זענען ספעציפיש, ווייל זיי פירן די פראגע צו דעם ארט וואו זיי קענען צוקומען צו דער אינפארמאציע און זי ניצן. די וואַרענונג: ניצט זיי מיט פארזיכטיגקייט, און דריקט אויס אייערע מיינונגען און אידייען אין א קלארער שפראך, דאס וועט העלפן.

סיכום-פונקטן פון דעם טייל (9:36:31)

דער קורס שטעלט אויס די הויפט-לעקציעס פון דעם אפשניט:

- GenAI איז א טייל פון קינסטלעכער אינטעליגענץ, וואס ניצט טעכניקן פון מאשין-לערנען און דיפ-לערנען צו שאפן דאטא.
- LLMs קען מען צעטיילן אויף עטליכע אופנים, לויט זייערע פונקציעס און באניצערס, ד"ה לויט די פארשידענע געצייג און פעלדער אין וועלכע מען ניצט זיי.
- פראמפטס פירן דעם GenAI צו פארשטיין דעם הינטערגרונט און די ספעציפישע פאדערונגען פון דער אויפגאבע; מיט פראמפטס רעדט מען צו די LLMs.
- GenAI טוט פארשידענע אויפגאבן, ווי שאפן א רעזומע און LinkedIn -פראפילן, קלאסיפיצירן אימעילס, sentiment analysis, און טאג-טעגליכע זאכן. מען האט געוויזן ווי עפעקטיוו עס איז פאר LinkedIn-פראפילן, פאסטן, Instagram-פאסטן, מעסעדזשעס און בלאגס.
- multimodal פעאיגקייטן: מען קען אין דער זעלבער צייט פראצעסירן און שאפן פארשידענע סארטן דאטא, ווי טעקסט, בילדער און ווידעאס.

אריבער צו "אפטימיזירן GenAI מאדעלן" (9:38:04)

דאס נעקסטע טעמע איז "optimizing GenAI models". עס ווערט איבערגעחזרט אז GenAI שאפט פארשידענע סארטן אינהאלט, ווי בילדער, טעקסט און מוזיק, וואס ווייזט זיין ראלע אין שעפערישע ארבעטן. LLMs ווי ChatGPT זענען פארגעשריטענע AI מאדעלן וואס גייען ווייטער ווי בלויז פארשטיין מענטשליכע שפראך. דער קורס זאגט צו אז אין די קומענדיגע לעקציעס ווערט מען אויך ארומקוקן אנדערע מאדעלן אחוץ ChatGPT, ווי Gemini און Claude.

פראמפט-אינזשעניערינג: קלארע אנווייזונגען פאר AI (9:39:08)

פרייע און באצאלטע מכשירים, און API-אריינלייגן אין קאוד (9:39:08)

דער קורס דערמאנט אז די פארשידענע טשעט-מכשירים, ווי ChatGPT און DeepSeek, האבן טייל פרייע מעגליכקייטן און טייל באצאלטע. עס ווערט צוגעזאגט אז שפעטער ווערט מען אויך ערקלערן דעם פראקטישן צד: ווי אזוי מען ניצט די מכשירים אינעם אייגענעם קאוד.

דער געדאנק איז אזוי: אויב מען וויל בויען א קאוד אדער א פלאטפארמע וואס ניצט א גענעראטיוון AI-מכשיר, קען מען באקומען אן API פון OpenAI און שרייבן א Python קאוד וואס בויט אן אייגענע גענעראטיווע אפליקאציע, ענליך צו ChatGPT. די פלאטפארמע ארבעט אזוי:

- דער באניצער שרייבט עפעס אין דער פלאטפארמע.
- אינעווייניג רופט דער קאוד דעם API פון OpenAI.
- OpenAI שיקט צוריק א תשובה.
- די פלאטפארמע ווייזט די תשובה פארן באניצער.

עס ווערט געזאגט אז, אויב עס ווערט זיין צייט, ווערט מען אפהאלטן א באזונדערע לעקציע ווענן ארבעטן מיט API.

דער ציל: עפעקטיווע פראמפטס און אוואנסירטע טעכניקן (9:40:10)

פריער האט מען געזען ווי א סיכום (summarization) ווערט געמאכט דורך שרייבן פראמפטס. צום סוף פון דער לעקציע איז דער ציל צו לערנען ווי אזוי צו שרייבן עפעקטיווע פראמפטס און פארבעסערן די שמועס מיט גענעראטיווע מאדעלן.

דער קורס באטאנט אז די גרעסטע שווערקייט איז ווי אזוי צו שרייבן דעם פראמפט. יעדעס פרטל אינעם פראמפט איז וויכטיג. די פריערדיגע פראמפטס האבן געגעבן די ריכטיגע אינפארמאציע, אבער די פראגע איז: וועלכע ווענן און וועלכע מסגרות (frameworks) זענען דא צו שרייבן גוטע פראמפטס?

דערפאר ווערט מען לערנען אוואנסירטע טעכניקן וואס העלפן ארויסברענגען דעם געוואונטשענעם רעזולטאט פון די גענעראטיווע מאדעלן:

- Zero-shot
- Few-shot
- Chain of Thought
- Self-Consistency

- Tree of Thought

מיט די טעכניקן קען מען פארבעסערן דעם רעזולטאט לויט דער באזונדערער נוצונג.

פיין-טיונען דעם פראמפט: א קרייז פון פרובירן און פארבעסערן (9:41:43)

דער קורס רופט דעם פראצעס "פיין-טיונען" (fine-tuning) פון די פראמפטס, דאס הייסט שטיקווייז פארבעסערן דעם פראמפט. עס ווערט ערקלערט מיט א ביישפיל:

- מען האט אן LLM. מען שרייבט א פשוטן פראמפט P1 און מען גיט אים צום LLM.
- דער LLM גיט ארויס א רעזולטאט O1.
- דער מענטש קוקט און זעט אז דער רעזולטאט פאסט נישט פונקטליך צו די באדערפענישן.
- דורך א "feedback" מעכאניזם ענדערט מען דעם פראמפט אביסל, מען לייגט צו קלארערע אנווייזונגען, ספעציעל די וואס האבן געפעלט.
- מען גיט עס ווידער צום LLM און באקומט רעזולטאט O2.
- מען קוקט צי דאס איז וואס מען דארף. אויב נישט, פארבעסערט מען דעם פראמפט נאכאמאל.

דאס רופט מען "instruction fine-tuning": מען פארבעסערט די אנווייזונגען לויט דער נוצונג, ביז מען באקומט דעם געוואונטשענעם רעזולטאט. עס ווערט געווארנט אז דער מאדעל קען "hallucinate", דאס הייסט אויסטראכטן זאכן, אדער געבן אומקלארע תשובות. דעמאלט מוז מען ענדערן דעם פראמפט און פרובירן ווידער און ווידער. דאס איז בעצם א פראצעס פון פרובירן און טועה זיין (trial and error).

ריכטיגע אנווייזונגען, אדער א סטרוקטורירטער אינפוט און סטרוקטורירטער אויספוט, קענען געבן א שיינעם רעזולטאט. דאס האט מען שוין פריער געזען: וועו מען האט געגעבן גאר קלארע און פונקטליכע אנווייזונגען, האט מען באקומען שיינע מוסטערן (templates) פאר UI-דיזיינס.

וואס א גוטער פראמפט דארף האבן: קלארקייט, און דער ביישפיל פונעם קליינעם קינד (9:43:15)

פראמפט-אינזשעניערינג באדייט צו ניצן שמועס-AI מאדעלן, ווי ChatGPT, כדי צו ערהאלטן דעם געוואונטשענעם רעזולטאט. אלע טשעט-באטס זענען אזעלכע שמועס-AI מכשירים.

א גוט-געבויטער פראמפט מוז האבן קלארקייט (clarity), אין טעקסט. מען האט אפשר א קלארן בילד אינעם קאפ: "איך וויל מאכן אט די אפליקאציע, איך וויל אט אזא רעזולטאט". אבער דאס מוז מען איבערגעבן אין א קלארן טעקסט.

דער קורס ברענגט א משל: אויב מען וויל שיקן א צוויי-יאריג קינד אין געשעפט קויפן נאש, ווי אזוי וואלט מען עס אנגעוויזן? אט אזוי דארף מען אויך שרייבן א פראמפט פארן LLM. מען וואלט געזאגט:

- גיי ארויס פון שטוב.
- גיי בערך 50 מעטער.
- דארט זעסטו א דריי אויף לינקס. גיי נאך 10 מעטער.
- דארט זעסטו צוויי געשעפטן. איינע פון זיי הייסט, למשל, "grocery store".
- גיי אריין אין יענעם געשעפט, קויף נאש, למשל א זיסוואַרג פאר פינף רופיס, און גיב דעם קרעמער דאס געלט אויב ער בעט.

- קום צוריק די זעלבע שטרעקע וואס דו ביסט געגאנגען.

דאס איז א סעט פון קלארע אנווייזונגען, מיט פונקטליכע אינפארמאציע. מען זעט אינעם פראמפט קלארקייט וועגן דעם ציל, און ספעציפישקייט (specificity): גיי לינקס, גיי רעכטס, 10 מעטער, פינף רופיס, א זיסוואַרג, קום צוריק אהיים.

פארמאט און קאנטעקסט (9:45:57)

דער צווייטער עלעמענט איז דער פארמאט. דער רעזולטאט קען זיין אין טעקסט-פארמאט, JSON-פארמאט אדער XML-פארמאט. דאס זענען די מערסט באניצטע פארמאטן, הגם עס זענען פאראן נאך פיל. מען קען אויך ארויפלאדן א CSV טעקע, אדער א PDF. דער פארמאט מוז זיין ריכטיג.

דער דריטער עלעמענט איז קאנטעקסט. מען מוז געבן דעם LLM דעם נויטיגן קאנטעקסט כדי די קאמוניקאציע זאל זיין עפעקטיוו.

צום פארגלייך: מען קען אויך פשוט זאגן "ברענג מיר א זיסוואַרג פאר פינף רופיס", אבער פון וואנעט? דאס איז אן אומקלארע, אפענע פראגע.

ביישפיל: פלאנירן א רייזע (9:47:02)

נאך א ביישפיל: אויב מען שרייבט "איך וויל פארן איבער דער וועלט", וועט דער LLM פרעגן: וואוהין ווילסטו פארן? האסטו א באשטימטן פלאן אדער רייזע-פלאן (?itinerary דערפאר מוז מען אים אליין דערציילן די פרטים. א קלארער פראמפט וואלט געקלונגען אזוי:

- איך פלאן א רייזע קיין USA אויף פינף-זעקס טעג.
- איך פאר פון אינדיע.
- איך האף צו כאפן א פליגער אינדערפרי, כדי צו אנקומען דארט אין אוונט.
- איך וויל באזוכן צוויי שטעט אין א פינף-טאגיגער רייזע.
- איך פאר מיט דער פרוי און די קינדער.

אזוי שרייבט מען א גוט-אויסגעארבעטן פראמפט: פונקטליך, ספעציפיש און קלאר. דער כלל וואס דער קורס לערנט: וועג מען שרייבט א פראמפט, זאל מען זיך פארשטעלן אז מען רעדט מיט א צוויי-יאריג קינד וואס ווייסט גארנישט. אויב מען גיט באדייטנדיגע און קלארע אנווייזונגען, איז די פינקטלעכקייט פונעם רעזולטאט גאנץ וועניגער וויכטיג.

פארוואס פראמפט-אינזשעניערינג איז וויכטיג, און "גארדרעילס" (9:48:34)

דער קורס רעכנט אויס פארוואס פראמפט-אינזשעניערינג איז נויטיג:

- ארויסנעמען דעם מאקסימום פונעם AI.
- מוטיגן AI-וויסן (AI literacy) און שעפערישקייט.
- פארבעסערן די שמועס מיטן באניצער.
- פארגרעסערן עפעקטיווקייט.
- שטיצן אן עטישע באנוצונג פון AI.

וועגן עטיק ווערט ערקלערט: אויב מען בעט פון AI א רעצעפט ווי צו מאכן א באמבע, וועט עס נישט ענטפערן. אזעלכע זאכן ווערן אויסגעפילטערט דורך "guardrails" (שוץ-געלענדער). וועו מען גיט אריין א טעקסט, באשטימט די סיסטעם צי די אינפארמאציע איז נויטיג און צי זי איז עטיש פאר דער געזעלשאפט. פרעגט מען צום ביישפיל א רשימה פון אלע באשטאנדטיילן פון א באמבע, וועט עס נישט ענטפערן, ווייל די גארדרעילס ארבעטן אינעווייניג אין די סיסטעמען און באגרענעצן אזעלכע אינפארמאציע.

פארוואס דארף מען עס? ווען מען האט געשטעלט דעם מאדעל די דאטא, האט קיינער נישט נאכגעקוקט און באשטעטיגט יעדעס שטיקל. עס זענען געווען אומגעהויערע מאסן, טעראבייטס אויף טעראבייטס. דערפאר, ווען מען גיט אן אינפוט, קען דער מאדעל אפשר יא געבן אזא רעזולטאט.

די גארדרעילס שטייען ביי ביידע זייטן: ביים אינפוט און ביים אויספוט. אויב א פראגע שליפט דורך דעם אינפוט-גארדרעיל, וועט דער אויספוט-גארדרעיל עס אפשטעלן. אזוי דערגרייכט אומעטיגע אינפארמאציע נישט דעם רעזולטאט.

עטיק, שעפערישקייט, און AI אלס "co-pilot" (9:50:36)

דער וויכטיגסטער באגריף איז צו ניצן AI אויף אן עטישן אופן, אז עס זאל העלפן דער געזעלשאפט און זי נישט צעשטערן. דערצו וויל מען מוטיגן AI-וויסן און שעפערישקייט.

עס ווערט דערמאנט א פראגע פונעם פריערדיגן טאג: צי AI שעדיגט נישט די מענטשליכע שעפערישקייט? דער ענטפער איז: יא, עס שעדיגט, אבער מען דארף זיך נישט פולשטענדיג פארלאזן אויף די מכשירים. א טייל פון דער שעפערישקייט בלייבט ביים מענטש, און די מכשירים דינען ווי א "co-pilot" (מיט-פילאט). מען בעט זיי הילף אין געוויסע געביטן, זיי געבן ארויס א רעזולטאט, און מען קוקט עס איבער און ניצט עס אין די אייגענע אפליקאציעס.

פראמפט-ביישפילן: פשוט קענן אויסגעארבעט (9:51:36)

צום סוף ווייזט דער קורס ביישפילן. א פשוטער פראמפט: "Write a catchy caption for my Instagram post", דאס הייסט שרייב מיר א כאפנדיגע אונטערשריפט פאר מיין Instagram פאסט. אבער וואס הייסט "catchy"? מען האט עס נישט ערקלערט. ChatGPT האט געענטפערט לויט זיין אייגענער מיינונג, עפעס ווי "Embracing the beauty of simplicity in a chaotic world… simple joys… find yourself". לויט די דאטא וואס מען אים געשטעלט, האלט ChatGPT אז אזא טעקסט איז "catchy".

דערנאך קומט אן אויסגעארבעטער פראמפט מיט קלארע אנווייזונגען, וואס בעט:

- שרייב א קורצע און אנציענדיגע (concise and engaging) אונטערשריפט פאר א Instagram פאסט וועגן א געוויסן ערפרישונג-פראדוקט. דער פראמפט צילט שוין אויף דעם באשטימטן פראדוקט.
- דער ציל-עולם פאר דעם פאסט: יונגע ערוואקסענע, רייזנדע, און מענטשן וואס גייען אויף שמחות. אזוי וויסט דער מאדעל וועלכן סארט קונים מען צילט.
- לייג צו פינף "hashtags" וואס פארמאגן הויכע רעזולטאטן און זענען פארבונדן מיט דעם פראדוקט.

דער קורס פארגלייכט: דער ערשטער איז א פשוטער פראמפט, און דער צווייטער איז אן "engineered prompt", א פראמפט וואס מען האט אויסגעארבעט און צוגעלייגט פרטים.

פארגעשריטענע פראמפט-טעכניקן: zero-shot, few-shot און chain of thought

(9:53:45)

פון א פשוט'ן פראמפט צו אן "אינזשענירטן" פראמפט (9:53:45)

דער קורס הייבט אן מיט דערמאנען דעם הויפט-געדאנק פון פראמפט-אינזשעניריע: מען וויל נישט שרייבן סתם א פראמפט אויף גערוואָרטנס. מען וויל שרייבן אן "אינזשענירטן" פראמפט, וואס איז אזוי געבויט אז ער גיט ארויס פונקט די רעזולטאטן וואס מען דארף. דערפון גייט מען איבער צו די פארגעשריטענע טעכניקן פון פראמפטן.

עס ווערט ערקלערט אז עס זענען דא עטליכע באקאנטע יסוד-טעכניקן:

- zero-shot prompting – א פראמפט אן קיין שום ביישפיל.
- few-shot prompting – א פראמפט מיט עטליכע ביישפילן.
- chain of thought prompting – צעטיילן די ארבעט אין א קייט פון שריט.
- self-consistency prompting – זיך פארזיכערן אז מען באקומט פעסטע און גלייכע ענטפערס.
- tree of thought prompting – א טעכניק וואס צווייגט זיך אויס ווי א בוים.

חוץ די דאזיגע זענען דא נאך טעכניקן. איינע הייסט ReAct, דאס הייסט Reason and Act. דער געדאנק דערביי: דער מאדעל זאל קודם קלערן ("reason") איבער די אנווייזונגען וואס מען גיט אים, און נאכדעם טאן ("act") לויט יענע אנווייזונגען.

עס ווערט אויך דערמאנט א framework וואס הייסט CAR, א צוזאמשטעל פון "clue" און "reasoning". דא נעמט מען קודם ארויס די סימנים (clues) און די שכל-לויפן (reasoning), און ערשט דערנאך פיטערט מען דאס אריין.

לויט דעם קורס זענען דא נאך אסאך אנדערע s-frameworks, באזירט אויף פארשידענע וויסנשאפטלעכע ארטיקלען, וואס באניצער האבן געגוצט ביי פראמפט-טעכניקן. די יסוד-טעכניקן זענען אבער chain of thought און tree of thought, און אויף זיי וועט מען זיך קאנצענטרירן.

מען וועט זען ווי זיי העלפן ווען די רעזולטאטן זענען נישט אפטימאל. אין אזא פאל קען מען "טוויקן" און צופאסן דעם פראמפט, ביז מען באקומט די בעסטע רעזולטאטן.

zero-shot, one-shot און few-shot – וואס מיינען זיי (9:55:18)

דער קורס ערקלערט די נעמען פשוט:

- "zero" מיינט נול ביישפילן.
- "one" מיינט איין ביישפיל.
- "few" מיינט עטליכע ביישפילן.

דער ערשטער ביישפיל איז שאפן א פאסט פאר סאשעל-מידיא לטובת א פירמע (brand). ביי zero-shot גיט מען נישט קיין שום ביישפיל, נאר מען שרייבט פשוט: "איך וויל דזשענערירן א סאשעל-מידיא פאסט". דאס איז א פראמפט מיט נול ביישפילן.

ביי one-shot לייגט מען צו א ביישפיל. אויף דעם ווערט געגעבן א משל: מען טרעפט אויף Instagram א פאסט וואס מען האלט פאר זייער אינטערעסאנט און שעפעריש, און מען וויל שרייבן אן אייגענעם פאסט וואס זעט אויס ענליך. דאן לייגט מען יענעם פאסט אריין אין פראמפט און מען שרייבט: "איך וויל דזשענערירן א סאשעל-מידיא פאסט באזירט אויף דעם פאסט ווייטער אונטן".

דער אריגינעלער פראמפט בלייבט דער זעלבער, נאר מען לייגט צו ביישפילן:

- אויב מען גיט איין ביישפיל, איז דאס one-shot.
- אויב מען גיט מער ביישפילן, איז דאס few-shot.
- אויב מען גיט נישט קיין ביישפיל, איז דאס zero-shot.

פארוואס גיט מען ביישפילן, און וואס קאסט עס (9:58:25)

עס ווערט ערקלערט אז מיט ביישפילן גיט מען צו נאך context פאר GPT. מען זאגט אים כלומרשט: "איך וויל עפעס דזשענערירן, איך האב געזען עפעס אזוינס, שאף מיר אינהאלט באזירט אויף דעם ביישפיל". דער מאדעל קוקט דאן אויף דעם ביישפיל און דזשענערירט לויט אים.

וואס מער ביישפילן מען גיט, אלץ בעסער. אבער דער קורס ווארנט אז דאס "מער" קאסט טאקנס (tokens), ווי מען האט שוין פריער ערקלערט. מען קען געבן אזויפיל ביישפילן ווי מען וויל, אבער פאר יעדן ביישפיל צאלט מען אין טאקנס.

אין אלגעמיין זעט מען אז עס לוינט זיך צו ניצן one-shot אדער few-shot אנשטאט zero-shot. דער רעזולטאט קומט בעסער ארויס, ווייל מען דערקלערט דעם מאדעל קלארער וואס מען מיינט.

א משל: צוויי פריינד גייען קויפן א קלייד (9:59:27)

אום דאס צו דערקלערן ווערט געבראכט א משל פון צוויי פריינד וואס ווילן קויפן א קלייד.

דער ערשטער פרובירט צו ערקלערן פאר דעם צווייטן וועלכע קלייד ער וויל טראגן. דער צווייטער פרעגט: "האסטו א ביישפיל פון אזא קלייד?" דער ערשטער ענטפערט: "יא, אט אזוי זעט עס אויס". אצינד פארשטייט דער צווייטער אסאך בעסער אז זיין פריינד זוכט א קלייד וואס זעט אויס ענליך צו דעם מוסטער.

דאס זעלבע טוט מען ביים פיטערן אינפוט צו די LLM-ס. מען גיט א ביישפיל, למשל א בילד אדער א דיזיין וואס מען וויל, און מען בעט דעם מאדעל צו שאפן א רעזולטאט באזירט אויף יענעם דיזיין.

in-context learning – לערנען, נישט טריינירן (10:00:28)

דער קורס דעפינירט: די טעכניק גיט דעם מאדעל א באגרעניצטע צאל ספעציפישע ביישפילן אין א באשטימטן context.

מען שטרייכט שטארק אונטער אז דאס טריינירט נישט דעם מאדעל. מען גיט אים בלויז context מיט ביישפילן. ער נעמט די ביישפילן, פארשטייט פון זיי וואס מען וויל, און גיט ארויס דעם ריכטיגן סארט רעזולטאט.

דאס רופט זיך in-context learning. מען גיט דעם context אינעווייניג אינעם פראמפט, און פון דארט לערנט זיך דער LLM אליין פון די ביישפילן (ביישפיל 1, ביישפיל 2). די ביישפילן זענען א מין דאטא וואס מען גיט דעם מאדעל, און פון דער דאטא לערנט ער זיך "אינעם context".

אין אלגעמיין זענען in-context learning און געבן ביישפילן א בעסערער וועג צו פארבעסערן די ענויקייט פון רעזולטאט.

דער ביישפיל מיט די אויסגעטראכטע ווערטער (10:02:00)

ווייל ניט אלע האבן פארשטאנען, ווערט דאס איבערגעחזרט מיט א פראמפט. עס ווערט געבראכט אן אויסגעטראכט ווארט (עפעס ווי "plumboxy", וואס מען קען אפילו נישט ריכטיג ארויסרעדן). אין פראמפט שטייט אז דאס ווארט באדייט א פאנטאסטיש באשעפענישׂ וואס ווא־וינט אין א פארכישופטן וואלד. נאכדעם שטייט א ביישפיל פון א זאץ וואס ניצט דאס ווארט.

אזוי האט מען געגעבן סיי די דעפיניציע סיי א ביישפיל. דערנאך לייגט מען צו א צווייט אויסגעטראכט ווארט (עפעס ווי "quiz wizler") און מען בעט: "גיב מיר א ביישפיל פון א זאץ וואס ניצט דאס ווארט". ווייל מען האט געגעבן איין ביישפיל פריער, איז דאס one-shot.

דער מאדעל לערנט זיך פונעם ערשטן context און דזשענערירט לויט אים. ווען מען גיט מערערע ביישפילן, ווערט עס few-shot.

נאכאמאל: דריי וועגן צו שרייבן א פראמפט (10:03:35)

דער קורס חזר'ט איבער: מען גיט דעם LLM א פראמפט און ער גיט ארויס א רעזולטאט. ביים שרייבן דעם פראמפט זענען דא דריי שטייגערן: zero-shot, one-shot און few-shot.

- zero-shot איז דער פשוטסטער. מען וויל שרייבן אן Instagram פאסט לויט אן איידיע וואס מען האט אין קאפ, אן קיין ביישפילן, און מען באקומט א רעזולטאט.
- ביי one-shot לייגט מען צו א ביישפיל, למשל: "דאס איז געווען דער Instagram פאסט פון מיין פריינד וואס האט געשילדערט די און די אינפארמאציע, מיט די און די hashtags. דאס איז דער מוסטער וואס איך וויל". אן א ביישפיל וואלט דער מאדעל אליין אויסגעקליבן א מוסטער. מיטן ביישפיל באגרענעצט מען אים צום מוסטער וואס מען האט אין זינען.
- ביי few-shot גיט מען מער ביישפילן. איין ביישפיל קען ווייזן א געוויסן מוסטער, און א צווייטער ביישפיל עפעס אנדערש, למשל א שעפערישן טעקסט.

דער ציל: דער LLM זאל פארשטיין וואס מען טראכט אין קאפ, דורך די ביישפילן און דעם context. ער לערנט זיך פון זיי, אבער ער ווערט נישט טריינירט. דאס איז in-context learning.

(10:06:09) chain of thought prompting

די נעקסטע מעטאד איז chain of thought prompting. אנשטאט באקומען דעם גאנצן רעזולטאט מיט איינמאל, באקומט מען א סעריע רעזולטאטן איינס נאכן אנדערן. מען באקומט קודם א רעזולטאט, מען גיט אים ווייטער צום נעקסטן שריט, דערנאך צום נעקסטן, און אזוי ווייטער ווי א קייט.

דאס העלפט צו באקומען א פרטימדיגן ענטפער, ווייל מען קאנצענטרירט זיך קודם אויף איין טייל, און נאכדעם אויפן נעקסטן.

דער ביישפיל איז ווען די ביאלאגישע פארשידנקייט אויף דער וועלט (biodiversity):

- ערשטער פראמפט: "וואס זענען די הויפט-סכנות פאר דער גלאבאלער ביאלאגישער פארשידנקייט?" – דער מאדעל ענטפערט.

- צווייטער פראמפט: "לויט די סכנות, וועלכע שוץ-סטראטעגיעס (conservation) באווייזן זיך אלס די מערסטע עפעקטיוו?"

אזוי צעברעכט מען דעם פראמפט אויף טיילן. מען האט טאקע געקענט שרייבן ביידע פראגעס אין איין זאץ. אבער וועו מען ארבעט ערשט אויפן ערשטן טייל, קען מען נאכקוקן צי דער ענטפער פאסט פאר די באדערפענישן, און אויב נישט, קען מען דעם פראמפט ענדערן און צופאסן.

דער רעזולטאט פון איין פראמפט וועלכע דער אינפוט פארן נעקסטן. ביי יעדן שריט קען מען מאכן ענדערונגען און אפשאצן דעם ענטפער.

דער ביישפיל גייט וועלכע מיט א נעקסטן פראמפט: "וי פאסן זיך די סטראטעגיעס צו מיט די ארטיגע מענטשן-געמיינדעס?" דער ענטפער: מען זאל אריינציען די ארטיגע געמיינדעס אין באשלוס-מאכן, געבן אנדערע מקורות פרנסה, און פארשפרייטן לימוד און באוואוסטזיין. נאכדעם קומט נאך א פראגע: "וועלכע נייע טעכנאלאגיעס שטיצן די שוץ-באמיאונגען?"

דער עיקר פון chain of thought: מען גיט נישט דעם פראמפט ווי איין גאנצקייט. מען צעברעכט אים אויף טיילן און מען שאצט אפ יעדן טייל שריט נאך שריט.

self-consistency prompting – אנהייב (10:09:13)

צום סוף הייבט אן די ערקלערונג פון self-consistency. דער פראמפט גיט פעסטע, פארלעסלעכע און ענלעכע ענטפערס וועו מען פרעגט די זעלבע פראגע עטליכע מאל. דער ביישפיל: "וואס איז די הויפטשטאט פון פראנקרייך?" – דער מאדעל זאל דאורויף ענטפערן קאנסעקוועוענט דעם זעלבן ענטפער.

פראמפט-מעטאדן און מולטימאדאל פראמפטינג (10:09:45)

סעלף-קאנסיסטענסי און טרי-וו-טאט פראמפטינג (10:09:45)

דער קורס ענדיגט דא די ערקלערונג פון self-consistency prompting. דער ביישפיל: אויף דער פראגע וואס איז די הויפטשטאט פון פראנקרייך איז דער ענטפער Paris. מען וויסט אז דער ענטפער באזירט זיך אויף א פעסטן פאקט, און ער וועט זיך נישט טוישן. עס וועלט ערקלערט אז ביי געוועלכן טעקסט-גענערירן קען דער מאדעל אמאל געבן פארשידענע ענטפערס. ביי self-consistency prompting, וואו דער ענטפער איז פעסט, טוישן זיך די רעזולטאטן אבער נישט.

עס וועלט געזאגט אז מען קען דאס אליין אויספרובירן, און אז אלע די מעטאדן, ווי chain of thought, zero shot און one shot, וועלן וועלן דורכגעטון פראקטיש אין דער קומענדיגער לעקציע מיט ביישפילן. די טעאריע קומט קודם כדי דער לערנער זאל באקומען א טאפלטע חזרה: איינמאל אין טעאריע און נאכדעם אין פראקטיק.

א צווייטער ביישפיל: דער פראמפט "וואס איז דער גרעסטער פלאנעט אין אונזער זון-סיסטעם?" גיט דעם ענטפער Jupiter. אויב מען טוישט דעם פראמפט, למשל "קענסט מיר זאגן וועלכער פלאנעט איז דער גרעסטער אין אונזער זון-סיסטעם?", בלייבט דער ענטפער דער זעלבער. דער מאדעל פרובירט צו פארשטיין אויף וועלכן קאנטעקסט מען וויזט און וואס די פראגע מיינט, און די רעזולטאטן קומען ארויס כמעט די זעלבע.

דער לעצטער מעטאד איז tree of thought prompting. דער מאדעל בויט א בוים פון פארשידענע געדאנקען, וועגן אדער טריט, ענלעך צו די צווייגן פון א בוים. יעדער צווייג איז א מעגלעכער טריט אדער אידעע צו לייזן

דעם פראבלעם, און אלע טריט קענען צוהעלפן ביים לייזן. נאכדעם אפשאצט דער AI די טריט, באשליסט וועלכער גיט די בעסטע לייזונג, און גיט די לייזונג. מיט אנדערע וועגן: דער מאדעל צעטיילט א קאמפליצירטע אויפגאבע אויף עטלעכע טיילן, קוקט וועלכער טייל פאסט בעסטן צום פראבלעם, אדער פאראייניגט עטלעכע טריט אין איין ענטפער.

דער ביישפיל אויפן עקראן איז די גלייכונג x(x+3)=1. דער מאדעל לייזט זי טריט נאך טריט: טריט איינס, צוויי, דריי און פיר. מען קען עס אויך באצייכענען ווי "טראכטן און האנדלען", אדער reasoning. דער מאדעל צעברעקלט דעם פראבלעם אויף קלענערע טריט, אפשאצט זיי, און קומט אן צו דער ריכטיגער לייזונג.

אריינפיר צו מולטימאדאל פראמפטינג (10:12:47)

דא הייבט זיך אן א נייע לעקציע: ווי אזוי AI-מאדעלן קענען פארשטיין און פראצעסירן עטלעכע סארטן אינפוט אויף איינמאל, ווי טעקסט, בילדער און קול. מולטימאדאל פראמפטינג ווערט באשריבן ווי א מעטאד וואס טוישט דאס בילד פון קינסטלעכער אינטעליגענץ. די לעקציע וועט ערקלערן וואס דאס איז, פארוואס עס איז וויכטיג, און ווי מען ניצט עס אין פארשידענע אינדוסטריעס.

דער סדר פון דער לעקציע:

- אריינפיר צו מולטימאדאל פראמפטינג
- די אנטוויקלונג פון פראמפטינג
- די הויפט-באגריפן
- געצייג און פלאטפארמעס וואס שטיצן מולטימאדאל פראמפטינג
- ניצונגען און אפליקאציעס
- ווי מען הייבט אן מיט מולטימאדאל געצייג
- פשוטע מולטימאדאל פראמפטס אין פראקטיק
- א טיפע, טריט-ביי-טריט בויאונג פון א פראמפט
- עקספערימענטירן מיט פארשידענע ניצונגען

וואס איז מולטימאדאל פראמפטינג? עס איז א מעטאד וואס דערלויבט AI-מאדעלן צו פארשטיין און ענטפערן אויף אינפוט פון מער ווי איין "מאדאליטעט", דאס הייסט א קאמבינאציע פון טעקסט, בילדער, קול אדערווידעא. אנשטאט צו לייענען בלויז טעקסט, קען דער מאדעל אנאליזירן בילדער, קולות און אנדערע פארמען דאטא, און דערמיט געבן רייכערע ענטפערס וואס נעמען בעסער אין באטראכט דעם קאנטעקסט. עס איז ענלעך צו דעם ווי א מענטש פראצעסירט אינפארמאציע.

פארוואס איז עס וויכטיג? די וועלט באשטייט נישט בלויז פון טעקסט. מען לעבט אין א סביבה פול מיט חושים, און מולטימאדאל AI העלפט מאשינען דאס צו פארשטיין. מען ניצט עס צו שרייבן באשרייבונגען פאר בילדער (image captioning), אנאליזירן ווידעאס, אינטעראקטיוו chatbots וואס זעען אויך בילדער, און אפילו אין מעדיצינישע דיאגנאזן, וואו מען פאראייניגט בילדער מיט נאטיצן. עס איז אויך זייער נוצלעך צו שאפן אינהאלט און צו געבן רעקאמענדאציעס.

די אנטוויקלונג פון פראמפטינג (10:15:22)

פראמפטינג האט זיך אנטוויקלט פון פשוטע טעקסט-אינפוטס ביז קאמפליצירטע מולטימאדאל אינטעראקציעס. היינטיגע מאדעלן פארשטייען און ענטפערן אויף קאמבינאציעס פון טעקסט, בילדער, קול און נאך, און דאס גיט א טיפערן קאנטעקסט. אנגעהויבן האט זיך פראמפטינג מיט טעקסט-מאדעלן ווי GPT-2, וועלכע האבן געענטפערט בלויז אויף געשריבענעם אינפוט און זיך באנוצט מיט NLP (natural language processing). מיט דער צייט האבן מאדעלן אנגעהויבן צו פראצעסירן בילדער, קול און אפילו ווידעא. אזוי זענען געבוירן געווארן אמת'דיגע מולטימאדאל סיסטעמען וואס פארשטייען און פארבינדן פארשידענע סארטן אינפארמאציע.

ווי אזוי פארשטייען די מאדעלן עטלעכע סארטן אינפוט? מאדערנע מולטימאדאל מאדעלן האבן ספעציעלע ארכיטעקטורן, ווי vision transformers און מולטימאדאל נויראל-נעצן, צו פראצעסירן די פארשידענע אינפוטס. מען טריינירט זיי אויף געפארטע דאטא, למשל בילדער מיט באשרייבונגען אדער ווידעאס מיט טראנסקריפטן. די מאדעלן לערנען זיך ווי די פארשידענע מאדאליטעטן זענען פארבונדן איינע מיט די אנדערע. דערמיט קענען זיי באקומען עטלעכע סארטן אינפוט און געבן ענטפערס וואס בויען א בריק צווישן די חושים.

cross-modal attention העלפט דעם מאדעל פארבינדן די רעלעוואנטע אייגנשאפטן פון די פארשידענע אינפוטס, און דאס פארבעסערט דעם קאנטעקסט און די גענויקייט. אין דעם יסוד ניצן די מאדעלן א געמיינזאמען embedding-רוים, וואו טעקסט און בילדער וועלן איבערגעטראגן אויף איין געמיינזאם פארשטאנד. למשל, א בילד פון א הונט און דאס וואלט "הונט" וועלן געשטעלט נאענט איינס צום אנדערן אין דעם רוים, און אזוי קען דער מאדעל טראכטן איבער פארשידענע פארמאטן אויף איין פאראייניגטן אופן.

הויפט-באגריפן: מאדאליטעטן, יונימאדאל קעגן מולטימאדאל (10:17:25)

מאדאליטעטן זענען די פארשידענע סארטן אינפארמאציע וואס מען קען געבן פאר AI: טעקסט, בילדער, קול אדער ווידעא. זיי העלפן דעם מאדעל פארשטיין זאכן אזוי ווי מענטשן פארשטייען, דורך עטלעכע חושים:

- טעקסט: געשריבענע שפראך, ווי פראמפטס אדער דאקומענטן
- בילד: וויזועלער אינהאלט, ווי פאטאס, טשארטס אדער צייכענונגען
- קול: אינפוט פון קלאנג, ווי רעדן אדער מוזיק
- ווידעא: אינהאלט וואס גייט לויט צייט, וואו בילד און קול זענען פאראייניגט

ביי unimodal prompting באקומט דער מאדעל איין סארט אינפוט, געווענליך טעקסט. ביי מולטימאדאל פראמפטינג קומען עטלעכע סארטן, למשל א פראגע צוזאמען מיט א בילד אדער א דיאגראם. דאס פאדערט פונעם מאדעל צו פאראייניגן פארשידענע שטראמען אינפארמאציע און צו מאכן פארבינדונגען צווישן זיי.

די הויפט-אונטערשיידן צווישן די צוויי:

- סארט אינפוט: unimodal האט איין סארט, געווענליך טעקסט; מולטימאדאל האט עטלעכע, ווי טעקסט, בילדער, קול א.א.וו.
- טיפקייט פון קאנטעקסט: unimodal איז באגרעניצט צו איין פארעם אינפוט; מולטימאדאל איז רייכער אין קאנטעקסט מיט פאראייניגטע אינפוטס.
- ניצונגען: unimodal איז נאר פאר טעקסט-ארבעט, ווי שרייבן; מולטימאדאל פאסט פאר פילע און קאמפליצירטע אויפגאבן, ווי בילד-אנאליז און speech-to-text.

• פארשטאנד: unimodal איז ליניאר און קאנצענטרירט זיך אויף שפראך; מולטימאדאל איז cross-modal און מער ענלעך צו א מענטש.

שווערigקייטן (10:19:59)

טראץ דעם פאטענציאל ברענגט מולטימאדאל פראמפטינג מיט זיך שווערigקייטן. מאדעלן קענען פאלש צוזאמפאסן די אינפוטס, ארויסגעבן "hallucinations" (אויסגעטראכטע ענטפערס), אדער זיין נייגעדיג צו דער שטארקסטער מאדאליטעט:

• פראבלעמען מיטן פארמאט: דער מאדעל קען פאלש פארשטיין שלעכט-פארמאטירטע אינפוטס.

• באגרעניצטע שטיצע: נישט יעדער מאדעל קען ארבעטן מיט יעדער מאדאליטעט.

• אומקלארע אינפוטס: אומקלארע אינפוטס קענען פירן צו אומגענויע ענטפערס.

• א שווערע קאמפיוטער-לאסט: עטלעכע מאדאליטעטן פאדערן מער רעסורסן.

דערצו פעלן נאך סטאנדארט-מאסשטאבן (benchmarks) צו מעסטן די מאדעלן, און צו פאראייניגן טריינירונג-דאטא פון פארשידענע פארמאטן קען קאסטן אסאך רעסורסן.

געצייג און פלאטפארמעס (10:20:31)

• ChatGPT with vision: א ווערסיע פון ChatGPT וואס פארשטייט און אנאליזירט סיי טעקסט סיי בילדער. עס פאסט צו שרייבן באשרייבונגען פאר בילדער אדער צו ענטפערן אויף פראגעס וועגן וויזועלן אינהאלט. ChatGPT איז א chatbot פון OpenAI, געבויט אויף GPT (generative pre-trained transformer) מאדעלן. עס פארשטייט און שרייבט טעקסט ווי א מענטש: שרייבן, קאודירן, סיכומים, לערנען, און געווענליכע שמועסן. היינט האט עס שוין מולטימאדאל מעגלעכקייטן, און מען ניצט עס ברייט אין חינוך, ביזנעס און שעפערישער ארבעט. עס פאראייניגט בילד-דערקענונג מיט טעקסט-שרייבן: עס פראצעסירט דאס בילד, פארבינדט עס מיטן טעקסט, און גיט א פולשטענדיגן ענטפער.

• Gemini פון Google: א משפחה פון פארגעשריטענע AI-מאדעלן, געבויט צו קאנקורירן מיט GPT. עס איז מולטימאדאל: עס ארבעט מיט טעקסט, בילדער, קאוד און נאך. עס טרייבט געצייג ווי Bard און Google Workspace AI, און דער ציל איז צו געבן געוויע הילף איבער דער גאנצער Google-סיסטעם. עס העלפט ביי קאמפליצירטע אינטעראקציעס, ווי זוכן לויט בילדער און שאפן מולטימאדאל אינהאלט. עס פראצעסירט אינפוטס גאר שנעל, און ניצט פארגעשריטענע "fusion" מאדעלן צו פאראייניגן טעקסט, בילד און וויידעא און צו פארשטיין די פארבינדונגען צווישן זיי.

• Claude פון Anthropic: א AI וואס לייגט דעם טראפ אויף זיכערקייט, נוצלעכקייט און דורכזיכטיגקייט. עס איז באקאנט פאר באטראכטע ענטפערס, און מען קען עס גוט וועגדן לויט דעם וואס דער באניצער ויל. עס פאראייניגט טעקסט, בילדער און דאקומענטן פאר א טיפע אנאליז. עס איז א שטארקער קאנקורענט צו ChatGPT און איז שטארק אין PDF-ארבעט, פראגעס און ענטפערס, און סיכומים. עס צעטיילט גוט יעדע מאדאליטעט באזונדער, פראצעסירט זיי, און פאראייניגט זיי נאכדעם פאר אויפגאבן ווי דאקומענט-אנאליז אדער שאפן אינהאלט פון עטלעכע מקורות.

• E·DALL פון OpenAI: א מאדעל וואס שאפט בילדער פון הויכער קוואליטעט פון טעקסט-באשרייבונגען. עס איז באקאנט פאר דעטאלירטע, שעפערישע רעזולטאטן, און איז אריינגעבויט אין ChatGPT פאר באניצער אויפן Plus-פלאן. עס נעמט א טעקסט-פראמפט און איבערזעצט אים אין א פאסיגן בילד. עס האט אסאך געצייג צו בויען א סצענע, און מאכט פון איר א בילד, פון רעאליסטיש ביז פאנטאזיע.

ניצונגען און אפליקאציעס (10:24:38)

- באניצער קענען ארויפלאדן בילדער ווי דיאגראמען און טשארטס, און פרעגן פראגעס דערוועגן; דער AI גיט ענטפערס וואס פאסן צום קאנטעקסט.
- שאפן אינהאלט: AI שרייבט ארטיקלען, פאסטס פאר סאשעל מידיע אדער מארקעטינג-טעקסט, באזירט סיי אויף טעקסט סיי אויף בילדער.
- צייכענען פראדוקטן: e-commerce פלאטפארמעס ניצן בילדער און באשרייבונגען צו שאפן געוויע פראדוקט-צייכנס (tags) און רעקאמענדאציעס.
- אינטעראקטיוו לערנען: AI אנאליזירט בילדער, ווידעאס און טעקסט צו שאפן אינטעראקטיווע לערן-מאטעריאלן.
- צוטריטלעכקייט: עס העלפט באניצער וואס זענען שוואך, דורך איבערמאכן וויזועלן אינהאלט, ווי פאטאס און אינפאגראפיקס, אין באשרייבנדיגן טעקסט.

ווי מען שטעלט אויף א סביבה, און א דעמא (10:25:41)

די טריט צו אויפשטעלן א פשוטע סביבה פאר מולטימאדאל פראמפטינג:

- קלייבן א פלאטפארמע: זיך איינשרייבן ביי ChatGPT אדער Gemini.
- אינסטאלירן די נויטיגע געצייג און libraries.
- באקומען API keys פון דער פלאטפארמע און אויפשטעלן די סביבה.
- זיכער מאכן אז די אינפוטס זענען ריכטיג פארמאטירט און פאסן פאר דער אפליקאציע.
- אויספרובירן מיט פשוטע פראמפטס אדער פשוטע מולטימאדאל אויפגאבן.

וואס מען דארף: API-צוטריט, עטליכע Python libraries און א Jupyter notebook. מען קען ניצן cloud-פלאטפארמעס, הגם א טייל פון די אפליקאציעס קען מען לויפן אויפן אייגענעם קאמפיוטער. די אינפוט-פארמאטן: טעקסט, בילד, קול און וויידעא.

דערנאך הייבט זיך אן א דעמא מיט ChatGPT צו ווייזן ווי LLMs ארבעטן אויף מולטימאדאל אויפגאבן. עס ווערט באמערקט אז רוב LLM-געצייג, ווי Gemini און Claude, גיבן ענלעכע מעגלעכקייטן. פילע פון די מעגלעכקייטן זענען בנמצא נאר אין ChatGPT Pro; אין דער פרייער ווערסיע זענען זיי בנמצא נאר פאר א באגרעניצטע צאל פראמפטס, און פאר דער פולער מעגלעכקייט דארף מען זיך אבאנירן אויף Pro.

ערשטנס, נאכן זיך אריינלאגירן אין ChatGPT, דארף מען זיך פארזיכערן אז GPT-4 איז אקטיווירט. מען זעט א פלוס-קנעפל צו צולייגן מעדיע, אבער פריער ווערט געווויזן ווי מען שאפט אינהאלט מיט טעקסט-פראמפטס. דער פראמפט: "שרייב א סיכום פון דריי זאצן פון דער מעשה פון Inception, ווי דו ערקלערסט עס פאר א קינד פון צען יאר." ChatGPT גיט א קורצן סיכום פון דריי זאצן פון דער מעשה פון Inception.

פון מולטימאדאל ChatGPT ביז LangChain (10:28:49)

א פאסטער פאר Inception אין קארטון-סטיל (10:28:49)

דער קורס פירט ווייטער מיט א דעמא פון ChatGPT . מען לייענט דעם ערשטן זאץ פון א באשרייבונג וואס ChatGPT האט פריער געשריבן: Inception איז וועגן א מאן מיטן נאמען Cobb, וואס גייט אריין אין מענטשנס חלומות כדי צו גנבענען זייערע סודות, ווי א שפיאן אין זייער שלאף.

דערנאך גייט מען א טריט ווייטער: מען בעט ChatGPT צו מאכן א פאסטער לויט דער באשרייבונג, אין קארטון-סטיל. מען פאסט אויף אז דער פראמפט זאל ווייזן אויף דער באשרייבונג פון אויבן. מען לויפט עס און מען ווארט עטליכע סעקונדעס.

עס ווערט געווארנט אז ChatGPT ארבעט אמאל אונטער א שווערן דרוק פון פילע פארלאנגען, און דאס מאכן בילדער קען דעמאלט נעמען צייט. ווען עס ענדיגט זיך, זעט מען אז ChatGPT האט געמאכט א פאסטער פון Inception אין קארטון-סטיל, פונקט לויט דער באשרייבונג.

אנאליזירן א בערזע-טשארט (10:29:20)

דער קומענדיגער טייל ווייזט ווי ChatGPT קען אנאליזירן בילדער. דער קורס זאגט אז די אויפגאבע איז זייער ניצלעך פאר קאמפליצירטע בילדער, ווי למשל בערזע-טשארטס.

- מען קליקט אויפן פלוס-צייכן און מען קלייבט "add photos and files".
- מען קלייבט אויס דאס בילד.
- מען שרייבט דעם פראמפט: "Analyze the support and resistance levels of the attached 4hour interval chart of Google", דאס הייסט: אנאליזיר די support- און resistance-שטאפלען אויף דעם צוגעלייגטן טשארט פון Google, וואס יעדער פונקט אויף אים שטייט פאר פיר שעה.

ChatGPT האט דעם טשארט אנאליזירט אין איין מאמענט. עס האט געגעבן עטליכע support-שטאפלען און עטליכע resistance-שטאפלען, יעדן מיט אן ערקלערונג.

ערקלערן א דיאגראם געצייכנט מיט דער האנט (10:30:20)

דער צווייטער ביישפיל פון בילד-אנאליז: מען קליקט ווידער אויפן פלוס-צייכן און מען לאדנט ארויף א טשארט וואס איז געצייכנט מיט דער האנט. דער פראמפט: "Explain the attached image of a diagram step by step", דאס הייסט: ערקלער דעם צוגעלייגטן דיאגראם שריט נאך שריט.

מען ווייזט דאס בילד און דערנעבן די ענטפער:

- "Start of the process": די פראגראם הייבט אן צו לויפן.
- שריט איינס, "input value": מען גיט אריין א ווערט.
- דערנאך: א קאנטראל צי עס איז דא א "stack overflow".
- ווייטער ערקלערט ChatGPT די איבעריגע שריט, לויטן דיאגראם.

ChatGPT האט אויך פארשטאנען דעם עיקר פונעם דיאגראם. עס איז א נארמאלע "stack push"-אפעראציע מיט א קאנטראל אויף overflow: זי לאזט נישט צולייגן דאטא ווען דער stack איז פול, און זי באשטעטיגט אז

די דאטא איז אריין וועג עס איז דא פלאץ. ChatGPT האט אויך פארגעלייגט צו געבן דעם זעלבן קאוד אין Python אדער C. דער קורס רופט דאס א גוטער ביישפיל פון טראכטן מיט בילד און טעקסט צוזאמען.

לייזן א מאטעמאטישע אויפגאבע פון א בילד (10:31:30)

יעצט מאכט מען די אויפגאבע שוועריר מיט א מאטעמאטישן פראבלעם. מען קליקט ווידער אויפן פלוס-צייכן, "add photos", און מען קלייבט דאס בילד. דער פראמפט: "What is the answer to the math problem in the attached image?", דאס הייסט: וואס איז די ענטפער פון דער מאטעמאטישער אויפגאבע אויפן צוגעלייגטן בילד? אויפן בילד שטייט "Find the sum of the series", דאס הייסט: געפין די סומע פון דער סעריע.

מען זעט אז ChatGPT שרייבט הינטער די קוליסן LaTeX- אדער Markdown-סינטאקס, כדי ארויסצווועזן ריכטיג די מאטעמאטישע צייכנס און די גריכישע אותיות. ChatGPT האט נישט נאר גענייזט די אויפגאבעס, נאר אויך געגעבן אנווייזונגען שריט נאך שריט ווי אזוי מען לייזט זיי. נאך מער: עס האט דערקענט אז אויפן בילד זענען דא צוויי אויפגאבעס, נישט בלויז איינע.

א דאטא-סאיענס פראצעס פאר דעם diabetes דאטא-סעט (10:32:32)

אין דעם לעצטן ביישפיל טוט מען פארקערט פון מולטימאדאל פראמפטן: אנשטאט צו געבן א בילד, בעט מען ChatGPT צו מאכן א קאמפליצירט בילד לויט א באשרייבונג. צום ערשט גיט מען דעם פראמפט: "Provide a detailed process for data science and feature engineering for the diabetes healthcare data set", דאס הייסט: גיב א פרטימדיגן פראצעס פון דאטא-סאיענס און feature engineering פאר דעם diabetes healthcare דאטא-סעט.

דער קורס ערקלערט אז דער דאטא-סעט איז באקאנט און מען קען אים געפינען אויף דער אינטערנעץ, דערפאר פארשטייט ChatGPT געוני וואס מען זוכט. די ענטפער ברענגט אנווייזונגען שריט נאך שריט, צוזאמען מיט Python קאוד צו מאכן די אנאליז: EDA, דערקענען outliers, multivariate אנאליז, און דערנאך feature engineering ווי מען האט געבעטן, ביז צום בויען דעם מאדעל און געבן פאסיגע מעסטונגען פאר evaluation. נאכדעם וואס מען האט באקומען די שריט אין טעקסט, וויל מען אז ChatGPT זאל בויען דערפון א דיאגראם.

וואס איז LLM אפליקאציע-אנטוויקלונג (10:33:35)

דא הייבט זיך אן אן אנדער טייל פונעם קורס. LLM אפליקאציע-אנטוויקלונג איז דער פראצעס פון בויען אפליקאציעס וואס ניצן גרויסע שפראך-מאדעלן, ווי GPT3 אדער GPT4, כדי צו פארשטיין, שאפן און פראצעסירן מענטשליכע שפראך. אזעלכע אפליקאציעס קענען טון אויפגאבעס וואס פאדערן פארשטיין שפראך: ענטפערן אויף פראגעס, קירצן א טעקסט, שאפן אינהאלט, און אפילו פירן א שמועס.

דער משל: מען האט א קלוגן געהילף אויפן טעלעפאן. מען פרעגט אים: "ווי וועט זיין דאס וועטער אין New York מארגן?" און אין עטליכע סעקונדעס גיט ער א קלארע ענטפער, ווי א מענטש. דאס איז א אפליקאציע וואס ארבעט מיט אן LLM. אבער עס איז נישט באגרעניצט צו וועטער; עס ווערן געבראנגט ביישפילן:

- קאסטומער-סערוויס chatbots: מען באזוכט אן e-commerce וועבזייטל און מען האט א פראגע וועגן א באשטעלונג. אנשטאט צו ווארטן אויף אן אגענט, באקומט מען גלייך אן ענטפער פון אן אויטאמאטישן chatbot. ער ענטפערט אויף פראגעס, לייזט פראבלעמען, און פירט אפילו דורך דעם קויף-פראצעס.

• שרייב-געהילפן: מכשירים ווי Grammarly אדער שרייב-געהילפן וואס ארבעטן מיט GPT . מען שרייבט אן email אדער א באריכט, און די אפליקאציע לייגט פאר פארבעסערונגען אדער ענדיגט אפילו די זאצן. ווי מען וואלט געהאט א וירטועלן רעדאקטאר בייזייט.

• מכשירים פאר קאוד: א פראגראמירער ווייסט נישט ווי צו שרייבן א געוויסע פונקציע. ער שרייבט "create a Python function that adds the two numbers", דאס הייסט: מאך א Python פונקציע וואס לייגט צונויף צוויי נומערן, און דער מכשיר שרייבט גלייך דעם קאוד. LLMs העלפן פראגראמירער ארבעטן גיכער, ווייל זיי לייגן פאר אדער שרייבן קאוד לויט אנווייזונגען אין פשוטער שפראך.

• אנאליז פון דאקומענטן: מען דארף לייענען א לאנגן לעגאלן קאנטראקט, אבער מען האט נישט קיין צייט צו גיין דורך יעדן סעיף. מען לאדנט אים ארויף אויף א מכשיר מיט אן LLM, און דער מכשיר מאכט א קיצור פון די הויפט-פונקטן. ער לייענט און פארשטייט דאקומענטן ווי א מענטש.

וואס איז LangChain און זיינע הויפט-טיילן (10:35:38)

LangChain איז א שטארקער framework וואס העלפט פראגראמירער בויען אפליקאציעס מיט גרויסע שפראך-מאדעלן ווי GPT4. ער לאזט גרינג פארבינדן LLMs מיט דאטא פון דער וועלט, מיט מכשירים און מיט דרויסנדיגע סיסטעמען, כדי צו שאפן מער אנטוויקלטע, אינטעראקטיווע און קלוגע אפליקאציעס. דער קורס פארגלייכט LangChain צו א געצייג-קאסטן פאר בויען קלוגערע אפליקאציעס מיט LLMs. ער טוט די פאלגנדע זאכן:

• ער מאכט גרינג צו ארבעטן מיט שפראך-מאדעלן.

• ער ברענגט דאטא פון דאקומענטן אדער דאטאבאזעס.

• ער אויטאמאטיזירט אויפגאבעס און באשלוסן.

• ער היט אפ דעם קאנטעקסט, אדער זכרון, אין א שמועס.

LangChain איז געבויט פון עטליכע הויפט-טיילן, וואס מאכן אים בייגעוודיג און שטארק:

• Models: מאדעלן וואס שאפן טעקסט אדער טוען שפראך-אויפגאבעס.

• Prompts: באשטימען ווי מען פרעגט דעם מאדעל א פראגע אדער בעט א פעולה.

• Chains: פארבינדן עטליכע אויפגאבעס אין א סדר, אדער אין א workflow.

• Memory: היט אפ פריערדיגע שמועסן פאר קאנטעקסט.

• Agents: באשליסן דינאמיש וועלכע פעולות צו טון.

• Tools: דרויסנדיגע סיסטעמען, ווי APIs און דאטאבאזעס, וואס די אגענטן קענען ניצן.

• Retrievers: זוכן און ארויסציען פאסיגע אינפארמאציע פון גרויסע דאטא-סעטס.

• Output parsers: מאכן פון דעם פרייען ענטפער פונעם מאדעל ניצלעכע, ארגאניזירטע דאטא.

די אלע טיילן ארבעטן צוזאמען, כדי צו שאפן קלוגע, בייגעוודיגע און מאדולארע אפליקאציעס מיט LLMs, וואס גייען ווייט איבער פשוט שרייבן טעקסט.

ווי אזוי LangChain העלפט בויען LLM אפליקאציעס (10:37:38)

דער קורס רעכנט אויס זיבן וועגן ווי LangChain העלפט:

• איינס, גרינגע איינפלעכטונג פון LLMs: מען שטעקט גרינג אריין מאדעלן ווי GPT4, GPT3.5 און אנדערע, אן קאמפליצירטע איינשטעלונגען און אן שרייבן אלעס פון אנפאנג. למשל, אויב מען וויל אז די אפליקאציע זאל ניצן GPT4 צו שאפן טעקסט אדער ענטפערן אויף פראגעס, קען מען דאס איינשטעלן מיט LangChain אין בלויז עטליכע שורות קאוד.

• צוויי, אויטאמאטיזירן עטליכע שריט מיט chains: אין דער אלטער אנטוויקלונג האט מען געדארפט מיט דער האנט צונויפשטעלן די שריט: פרעגן, פראצעסירן די ענטפער, און באהאנדלען די דאטא. LangChain לאזט קייטלען די פעולות צוזאמען. ביים בויען א chatbot קייטלט מען: נעמען דעם באניצערס אריינגאב, שיקן עס צום LLM, און צוריקגעבן די ענטפער צום באניצער. דאס מאכט די אנטוויקלונג גיכער און מיט ווייניגער טעותים.

• דריי, גרינג ארויסציען דאטא: וועז דער LLM דארף ציען אינפארמאציע פון א ריזיגן הויפן דאקומענטן, שטעלט מען איין retrievers וואס זוכן אין דאקומענטן אדער דאטאבאזעס און ברענגען צוריק די פאסיגסטע אינפארמאציע. למשל, אויב די אפליקאציע דארף ענטפערן אויף פראגעס וועגן דער פירמעס quarterly report, קען LangChain אויטאמאטיש ארויסציען די פאסיגסטע טיילן פונעם באריכט.

• פיר, זכרון פאר קלוגערע שמועסן: ביי chatbots און ווירטועלע געהילפן איז וויכטיג צו געדענקען וואס מען האט פריער געזעדט. LangChain לאזט צולייגן זכרון, אזוי אז דער LLM געדענקט וואס איז געזאגט געווארן און ענטפערט לויטן קאנטעקסט. למשל, פרעגט מען "וועגן וואס האבן מיר געזעדט נעכטן?", קען דער געהילף זיך צוריקוועגדן צום פריערדיגן שמועס.

• פינף, באשלוסן מיט אגענטן: אגענטן באשליסן אליין וועלכע פעולה צו טון, ווי פרעגן א דאטאבאזע, רופן אן API אדער שטעלן נאך א פראגע. למשל, פרעגט א באניצער "ווי איז דאס וועטער אין ?Paris", קען אן אגענט באשליסן צו ברענגען דאס וועטער פון א לעבעדיגן API, אזוי אז די ענטפער איז גענוי און לויט דער איצטיגער צייט.

• זעקס, איינפלעכטן מכשירים פאר נאך מעגלעכקייטן: LLM אפליקאציעס דארפן אפט ארבעטן מיט APIs, דאטאבאזעס אדער וועבזייטלעך. LangChain מאכט גרינג צו פארבינדן די מכשירים, אזוי אז דער LLM קען טון פעולות לויט פאקטישער דאטא. למשל, אין אן אפליקאציע פאר באשטעלן פליגער-בילעטן קען מען איינפלעכטן די APIs פון פליגער-באשטעלונגען, און דער LLM וועט קענען באשטעלן פאר באניצער גלייך אויפן ארט.

• זיבן, ער מאכט פשוט קאמפליצירטע אויפגאבעס: LangChain נעמט אויף זיך די שווערע טיילן, ווי קייטלען שריט, איינפלעכטן מכשירים און פירן זכרון, אזוי אז מען קען זיך קאנצענטרירן אויף דער הויפט-פונקציאנאליטעט. לויטן קורס איז דאס א גרויסער שינוי פאר פראגראמירער וואס ווילן בויען קלוגע אפליקאציעס מיט LLMs, אן בויען אלעס פון דער ערד.

פארוואס מען שטעלט LangChain פאר (10:41:12)

לויטן קורס איז LangChain געווארן דער ערשטער framework פאר פראגראמירער וואס ארבעטן מיט LLMs, ווייל ער מאכט די אנטוויקלונג גיכער, גרינגער און שטארקער. די סיבות:

• מאדולאריטעט און בייגעוודיגקייט: מען קלייבט נאר די טיילן וואס מען דארף, צי א מאדעל, צי א זכרון-סיסטעם, צי אן אגענט, און מען מישט זיי אן שוועריגקייטן. די מעלה: מען בויט נאר וואס מען דארף, און די אפליקאציע איז מער צוגעפאסט.

- גיכערע אנטוויקלונג און פראטאטיפן: פון א געדאנק צו אן ארבעטנדיגער אפליקאציע אין קורצער צייט, דאנק פארטיגע טיילן ווי chains, פראמפטס, אגענטן און זכרון. מען פרובירט אויס געדאנקען אן אויסטראכטן דאס ראד פון ס'ניי. די מעלה: מען שפארט צייט און מען אנטוויקלט די פונקציעס גיכער.
- איינגעבויטע מכשירים פאר דרויסנדיגע פארבינדונגען: APIs, דאטאבאזעס און דרויסנדיגע דאטא-קוואלן. צי מען דארף פרעגן א דאטאבאזע, ניצן א web search API , אדער ארבעטן מיט אייגענע מכשירים, האט LangChain דערפאר א לייזונג.
- זכרון פאר אפליקאציעס מיט קאנטעקסט: רוב LLMs האבן נישט קיין זכרון; זיי פארגעסן אלעס ווען דער שמועס ענדיגט זיך. LangChain לאזט אפהיטן און פירן זכרון, וואס איז נויטיג פאר chatbots און ווירטועלע געהילפן.
- אגענטן וואס באשליסן: זיי באשליסן לויטן אריינגאב צי צו פרעגן א דאטאבאזע, רופן אן API אדער בעטן נאך אינפארמאציע. דאס מאכט די אפליקאציע קלוגער און זעלבשטענדיגער.
- פשוטערע קאמפליצירטע workflows: מיט chains אויטאמאטיזירט מען פראצעסן פון עטליכע שריט, וואס פארבינדן פראמפטס, מאדעלן און מכשירים. דאס איז בפרט ניצלעך ווען די אפליקאציע דארף גיין לויט א סדר פון פעולות, ווייל מען דארף נישט צונויפנייען אלעס מיט דער האנט.
- אן אקטיווע קהילה און open source: אויף דעם פונקט הייבט מען אן צו רעדן ווען דער אפשניט ענדיגט זיך.

LangChain: מאדעלן, פראמפטס און די ערשטע קאוד (10:44:15)

פארוואס LangChain און די הויפט-טיילן (10:44:15)

דער קורס ענדיגט די ערקלערונג פארוואס LangChain איז נוצליך: עס איז open source, און עס האט אן אקטיווע קהילה פון פראגראמירער וואס פארבעסערן עס כסדר. דאס מיינט אז מען באקומט נייע פיטשערס, אפדעיטס און הילף פון אנדערע פראגראמירער. קורץ געזאגט, LangChain מאכט די אנטוויקלונג פון LLM-אפליקאציעס פשוט'ער, שנעלער און שטארקער. עס גיט די געצייג וואס מען דארף צו בויען קלוגע אפליקאציעס, אן דארפן שרייבן אלעס פון אנהייב.

דערנאך גייט מען איבער צום נעקסטן קאפיטל: די הויפט-טיילן (core components) פון LangChain, וואס ארבעטן צוזאמען.

מאדעלן (10:44:45)

דער ערשטער טייל איז מאדעלן. אין דער וועלט פון LLM-אפליקאציעס איז א מאדעל א קלוגע פראגראם וואס איז טרייניירט געווארן צו פארשטיין און ארויסגעבן מענטשליכע שפראך. די מאדעלן ווערן געבויט מיט ריזיגע מאסן דאטא, ווי ספרים, וועבזייטלעך און ארטיקלען, כדי צו לערנען ווי שפראך ארבעט: גראמאטיק, פאקטן, לאגישע טראכטונג און קאנטעקסט.

פשוט געזאגט, א מאדעל איז ווי א מוח וואס קען לייענען און שרייבן שפראך. מען קען אים פרעגן שאלות, געבן פראמפטס אדער אינפארמאציע, און ער ענטפערט אויף א שכל'דיגן אופן, לויט דעם וואס ער האט געלערנט.

דער ביישפיל: GPT-3 אדער GPT-4, צוויי באקאנטע שפראך-מאדעלן פון OpenAI. מען פרעגט: "וואס איז די הויפטשטאט פון פראנקרייך?" דער מאדעל פאראלעט די שאלה, פארשטייט אז זי האנדלט פון געאגראפיע, און דערקענט די ווערטער "הויפטשטאט" און "פראנקרייך" אין קאנטעקסט פון לענדער. דערנאך ענטפערט ער:

"די הויפטשטאט פון פראנקרייך איז פאריז." דער מאדעל, ווי GPT-4, האט געלערנט פון די טעקסטן אויף וועלכע ער איז טריינירט געווארן אז פאריז איז די הויפטשטאט, און ער ניצט זיין שפראך-פארשטאנד צו געבן א ענטפער ווי א מענטש.

פראמפטס (10:45:45)

ביי גרויסע שפראך-מאדעלן ווי GPT-4 אדער GPT-3, איז א פראמפט פשוט די אינפוט אדער אנווייזונג וואס מען גיט דעם מאדעל. עס איז ווי א שאלה אדער בקשה וואס פירט דעם מאדעל ביים ארויסגעבן א ענטפער. א פראמפט איז א וועג צו בעטן דעם מאדעל עפעס צו טון, צי צו ענטפערן א שאלה, צי צו פארענדיגן א זאץ, אדער אפילו צו שרייבן א מעשה.

צוויי ביישפילן:

- א פשוטע שאלה: "וואס איז די הויפטשטאט פון יאפאן?" דער מאדעל ענטפערט: "די הויפטשטאט פון יאפאן איז טאקיא."
- א שעפערישער פראמפט: "שרייב א קורצע מעשה וועגן א קאץ וואס האט ליב צו שווימען." דער מאדעל שרייבט א מעשה וועגן א קאץ וואס לערנט זיך שווימען און האט הנאה דערפון.

Output Parsers (10:46:47)

אן output parser איז א געצייג וואס העלפט פארארבעטן און ארגאניזירן דעם טעקסט וואס א שפראך-מאדעל גיט ארויס. דער ענטפער קען זיין א לאנגער פאראגראף, זאצן, אדער פרייער טעקסט, אבער אפט דארף מען ארויסנעמען ספעציפישע שטיקלעך אינפארמאציע דערפון, עס אויסרייניגן און פארמאטירן אין א נוצליכער פארעם. דאס אלעס וועט מען זען אין פראקטיק.

OpenAI API פרייזן (10:47:19)

וועגן די פרייזן פון דעם OpenAI API קען מען גיין גלייך אויפן זייטל פון OpenAI API pricing. וועט מען סקראלט אראפ זעט מען די דעטאלירטע פרייזן פאר אלע לעצטע מאדעלן: input, cached input און output.

אין דעם קורס ניצט מען GPT-3.5 Turbo. לויט דער לעקציע איז דער פרייז דערפאר 0.50 דאלער פאר input און 1.50 דאלער פאר) outputעס ווערן אויך דערמאנט צאלן פאר טריינירן, 3 און 8 דאלער). מען קען אויסקלייבן וועלכן API עס פאסט פאר דעם פראיעקט.

אריינלאדן דעם API שליסל (10:47:51)

די ערשטע קאוד איז גאנץ פשוט:

- מען אימפארטירט די ביבליאטעקן os און openai. העלפט ארבעטן מיט דער אפערירונג-סיסטעם, און דא ניצט מען עס צו כאפן environment variables ווי די API שליסלען. openai איז די ביבליאטעק וואס לאזט ארבעטן מיט די מאדעלן פון OpenAI.
- דערנאך ברענגט מען אריין די dotenv ביבליאטעק. זי איז זייער נוצליך, ווייל זי לאזט היטן סענסיטיווע אינפארמאציע, ווי API שליסלען, אין א .env טעקע, אנשטאט זיי שרייבן גלייך אין קאוד, וואס וואלט געווען זייער ריזיקאליש.

• די פונקציע find_dotenv זוכט די .env טעקע, און load_dotenv לאדט אירע אינהאלטן אין די environment. דאס איז וויכטיג, ווייל אזוי ווערט דער API שליסל פון דער טעקע צוגענגליך פאר דעם Python קאוד ווי אן environment variable.

• צום סוף נעמט מען דעם שליסל דורך os.environ['OPENAI_API_KEY'] און מען שטעלט אים אריין אין openai.api_key. אזוי ווערן אלע רופן צום API באשטעטיגט מיט דעם שליסל.

אויף דעם אופן קען מען ניצן OpenAI'ס סערוויסעס, ווי ארויסגעבן טעקסט, אן אנטפלעקן די סענסיטיווע שליסלען אינעם קאוד.

אויסקלייבן די מאדעל-ווערסיע לויטן דאטום (10:49:23)

דער נעקסטער קאוד באהאנדלט דאס וואס אן עלטערע LLM ווערט אויסגעשטעלט (deprecation), דורך קאנטראלירן דעם היינטיגן דאטום און באשטימען די מאדעל-ווערסיע לויט דעם:

• עס כאפט דעם היינטיגן דאטום מיט datetime.datetime.now().date() און היט עס אין דעם variable current_date.

• עס באשטימט א ציל-דאטום, דעם 12טן יוני 2024, נאך ווען דער מאדעל זאל איבערגיין צו א נייע ווערסיע.

• אויב דער היינטיגער דאטום איז נאכן ציל-דאטום, שטעלט מען דעם מאדעל אויף gpt-3.5-turbo. אויב מען איז נאך פאר דעם 12טן יוני 2024, ניצט מען ווייטער gpt-3.5-turbo-0301, אן עלטערע ווערסיע.

אזוי פאסט זיך דער קאוד אויטאמאטיש צו ווען עלטערע מאדעלן ווערן אויסגעשטעלט, און ער גייט איבער צו א נייערן ווען דער ציל-דאטום קומט.

די פונקציע get_completion (10:50:27)

די פונקציע get_completion נעמט א פראמפט און א מאדעל (דער דיפאָלט איז דער variable llm_model), און באקומט א ענטפער פונעם OpenAI API:

• ערשט פארמאטירט זי דעם פראמפט אין א מעסעדזש אין דער סטרוקטור וואס דער API פארשטייט, מיט דעם באניצער'ס מעסעדזש אלס content.

• דערנאך שיקט זי עס צום chat completion API מיט openai.ChatCompletion.create, און גיט אן דעם מאדעל, די מעסעדזשעס, און temperature פון 0, וואס מיינט או דער מאדעל וועט געבן מער באשטימטע (deterministic) ענטפערס, ווייניגער צופעליג.

• צום סוף גיט זי צוריק דעם ענטפער דורך נעמען די ערשטע ברירה (choice) פונעם רעזולטאט און ארויסנעמען דעם טעקסט.

מיט אנדערע ווערטער: די פונקציע שיקט דעם באניצער'ס פראמפט צום מאדעל און גיט צוריק דעם ענטפער.

ווען מען רופט get_completion("What is 1+1?"), שיקט די פונקציע דעם פראמפט צום מאדעל (דער דיפאָלט llm_model), דער מאדעל פאראַרבעט אים, און די פונקציע גיט צוריק דעם ענטפער. ווען מען לויפט עס, זעט מען: 1+1 איז 2.

א קלאגע-אימעיל און א סטיל (10:52:00)

דערנאך באקומט דער variable customer_email א טעקסט פון עטליכע שורות, מיט דריי געזופיסלעך (triple quotes). עס איז א שפאסיגער מעסעדזש פון א קונה, געשריבן אין "פיראטן-שפראך". דער קונה שרייבט זיין פראסטראציע: דער דעקל פון זיין בלענדער איז אראפגעפלויגן און האט געמאכט א באלאגאן, און ער איז אויפגערעגט אז די גאראנטיע דעקט נישט די קאסטן פון אויסריניגן. די backslashes ווערן געזוכט צו צעטיילן דעם טעקסט אויף עטליכע שורות כדי עס זאל זיין לייכטער צו לייענען, אבער עס בלייבט איין המשך'דיגער טעקסט.

דער variable style באקומט אויך א טעקסט, וואס באשרייבט א סטיל פאר שרייבן: אמעריקאנער ענגליש, אין א רואיגן און רעספעקטפולן טאן. אויך דא ניצט מען backslashes פאר לייענבארקייט. דער טעקסט קען שפעטער ווערן געזוכט צו באשטימען דעם טאן און סטיל פון די ענטפערס.

בויען דעם פראמפט און באקומען דעם ענטפער (10:53:34)

דער variable prompt ווערט געשאפן מיט אן f-string, וואס לאזט אריינשטעלן variables גלייך אין א טעקסט. דער פראמפט הייסט דעם מאדעל איבערזעצן דעם טעקסט וואס איז ארומגענומען מיט דריי backticks אין דעם סטיל וואס דער variable style באשטימט. דער customer_email ווערט אריינגעשטעלט צווישן די דריי backticks, וואס ווערן אפט געזוכט צו צייכענען קאוד אדער פאר-פארמאטירטן טעקסט. די f-string מאכט עס גרינג צו אריינשטעלן style און customer_email דינאמיש. print(prompt) ווייזט דעם גאנצן פראמפט.

דערנאך שרייבט מען:

get_completion(prompt) = response

דער פראמפט ווערט געשיקט צום OpenAI מאדעל, וואס פאראַרבעט אים לויט די אנווייזונגען. דער רעזולטאט ווערט געהיט אין response, וואס אנטהאלט די איבערגעזעצטע ווערסיע פונעם אימעיל אינעם געבעטענעם סטיל. מען זעט אינעם ענטפער: "I'm really frustrated that my blender lid flew off and splattered my kitchen walls", די זעלבע קלאגע, נאר אין א רואיגן, רעספעקטפולן ענגליש.

דאס זעלבע מיט LangChain (10:55:37)

יעצט ווייזט מען ווי מען טוט דאס מיט LangChain. ערשט לויפט מען pip install --upgrade langchain.

דערנאך אימפארטירט מען די קלאס ChatOpenAI פון langchain.chat_models. ChatOpenAI איז א קלאס פון LangChain וואס לאזט ארבעטן מיט OpenAI'ס מאדעלן אין א שמועס-פארמאט, און מאכט עס גרינגער אריינצוברענגען שמועס-AI אין אן אפליקאציע.

דערנאך שאפט מען א נייע instance פון ChatOpenAI מיט temperature=0.0 און model=llm_model. דער temperature פאראמעטער קאנטראלירט די מאס פון צופעליגקייט און שעפערישקייט אין די ענטפערס. מיט 0.0 זאגט מען דעם מאדעל צו געבן זייער פאראויסזעבארע, באשטימטע ענטפערס, כמעט אן צופעליגקייט, אזוי אז די ענטפערס זענען מער פאקוסירט און קאנסיסטענט. דער מאדעל איז וועלכער עס שטייט אין llm_model, למשל GPT-3.5 Turbo אדער אן אנדערע וועלכע.

א פראמפט-מוסטער (10:57:11)

דער variable template_string באקומט א טעקסט וואס בעט דעם מאדעל איבערזעצן דעם טעקסט צווישן די דריי backticks אין א געוויסן סטיל. דא זענען style און text פלאצהאלטערס (placeholders), וואס ווערן אויסגעפילט מיט די ווערטן וואס מען גיט צום מוסטער. אויך דא ניצט מען backslashes פאר לייענבארקייט, אבער עס בלייבט איין טעקסט. אזא מוסטער איז נוצליך ווען מען דארף שאפן פראמפטס מיט ספעציפישן אינהאלט און סטיל דינאמיש.

דערנאך אימפארטירט מען ChatPromptTemplate פון langchain.prompts, כדי צו שאפן א פראמפט-מוסטער פאר שמועס-אינטעראקציעס. די מעטאד from_template(template_string) נעמט דעם טעקסט מיט די פלאצהאלטערס און מאכט דערפון א מוסטער וואס מען קען ווידער ניצן און אויספילן מיט ספעציפישע ווערטן. דאס גיט א קאנסיסטענטן און פלעקסיבלן ווע צו שאפן פראמפטס, וואו מען קען לייכט צופאסן דעם טעקסט און סטיל פאר יעדן פאל.

צום סוף, prompt_template.messages[0].prompt נעמט די ערשטע מעסעדזש אינעם prompt_template אביעקט (דער אינדעקס 0 קלייבט אויס די ערשטע מעסעדזש פון דער ליסטע) און ברענגט ארויס איר פראמפט.

פראמפט-מוסטערן, אויסגאנג-פארזערס און זכרון (10:58:43)

דער פראמפט-מוסטער און זיינע פלאצהאלטערס (10:58:43)

דער קורס ערקלערט ווייטער ווי אזוי מען ארבעט מיט א פראמפט-מוסטער (prompt template) אין LangChain. דער אויסדרוק wie prompt ציט ארויס דעם פאקטישן פראמפט פונעם ערשטן מעסעדזש אין דעם מוסטער. אזוי קען מען זען דעם מוסטער, אדער אים טוישן, איידער מען ניצט אים צו שאפן א דינאמישן פראמפט פארן מאדעל. מען לויפט דעם קאוד און מען זעט דעם ערשטן פראמפט.

דערנאך ווערט ערקלערט די שורה prompt_template.messages[0].prompt.input_variables. זי נעמט ארויס די אריינגאנג-וואריאבלען פונעם ערשטן מעסעדזש. דער מוסטער איז געשאפן געווארן מיט ChatPromptTemplate, און יעדער מעסעדזש דערין איז א פראמפט מיט פלאצהאלטערס, ווי style און text. וועו מען שרייבט input_variables, באקומט מען די רשימה פון וואריאבלען-נעמען וואס דער מוסטער ניצט. אזוי וויסט מען וועלכע ווערטן מען דארף צושטעלן כדי צו פארענדיגן דעם פראמפט, איידער מען שיקט אים צום מאדעל.

א סטיל פארן קונה און א "פיראט"-בריוו (10:59:13)

דער וואריאבל customer_style באקומט א סטרינג פון עטליכע שורות. ער באשרייבט א געוויסן שרייב-סטיל: אמעריקאנער ענגליש, אין א רואיגן און רעספעקטפולן טאן. דער באקסלעש (\) אין סוף פון יעדער שורה לאזט צעטיילן דעם סטרינג אויף עטליכע שורות, כדי ער זאל זיין גרינגער צו לייענען. דער סטיל וועט גענוצט ווערן ביים רעדן מיט א קונה, למשל צו שאפן ענטפערס אויף אן איידעלן און פראפעסיאנעלן אופן.

דער וואריאבל customer_email באקומט אויך א סטרינג פון עטליכע שורות. דאס איז א שפילעוודיגער מעסעדזש פון א קונה, געשריבן אין דעם סטיל פון א ים-גזלן (פיראט). דא אויך צעטיילט דער באקסלעש דעם טעקסט אויף שורות צוליב לייענבארקייט, אבער אין דעם וואריאבל בלייבט עס איין גאנצער סטרינג. דער מעסעדזש איז געמאכט צו שפילן א הומאריסטישע קלאגע, צוליב טעסטן אדער דעמאנסטראציע.

פילן דעם מוסטער און איבערזעצן דעם סטיל (11:00:14)

דא ווערט גענוצט די מעטאד prompt_template.format_messages, וואס פילט אן די פלאצהאלטערס אין דעם מוסטער מיט באשטימטע ווערטן:

customer_messages = prompt_template.format_messages(style=customer_style, text=customer_email)

אזוי ווערט style פארביטן מיטן שרייב-סטיל, און text מיט דער קלאגע פונעם קונה. דער רעזולטאט, customer_messages, איז א פארמאטירטער מעסעדזש. די דינאמישע ווערטן זענען אריינגעשטעלט אין דעם מוסטער, און עס קומט ארויס א צוגעפאסטער פראמפט וואס מען קען שיקן צום שפראך-מאדעל. אזוי קען מען שאפן מעסעדזשעס וואס פאסן צום קאנטעקסט, לויט די אריינגאנג-וואריאבלען.

מען דרוקט אויס דעם טיפ פון customer_messages און מען זעט אז עס איז א list . דער ערשטער עלעמענט, customer_messages[0], איז פונעם טיפ langchain.schema.HumanMessage. מען קוקט אויפן ערשטן מעסעדזש פונעם קונה.

דערנאך רופט מען דעם גרויסן שפראך-מאדעל (LLM), כדי ער זאל איבערזעצן דעם סטיל פונעם מעסעדזש. דער ענטפער ווייזט דעם טויש: "Oh man, I'm really frustrated that my blender lid flew off and made a mess of kitchen", דאס הייסט דער קונה איז פראסטרירט, ווייל דער דעקל פון זיין בלענדער איז אראפגעפלויגן און האט פארשמירט די קיך. מען זעט קלאר דעם אונטערשייד צווישן דעם אריגינעלן פיראט-טעקסט און דעם רואיגן נוסח.

דער ענטפער פון קונה-סערוויס אין פיראט-טאן (11:01:48)

דער וואריאבל service_reply באקומט א סטרינג פון עטליכע שורות. ער שטעלט פאר דעם ענטפער פון קונה-סערוויס אויף דער קלאגע. דער מעסעדזש איז געשריבן אין א דירעקטן און אביסל לייכטן טאן, און ער ערקלערט אז די גראנטיע דעקט נישט די הוצאות פון אויסריניגן. דער קורס באמערקט אז דער ענטפער איז קלאר אבער אביסל שארף, און אז ער איז נאר א ביישפיל.

דער וואריאבל service_style_pirate באשרייבט א טאן פאר דער קאמוניקאציע: הויפלעך, אבער אויף א פיראט-אופן.

ווייטער ווערט ווידער גענוצט prompt_template.format_messages, דאס מאל פארן סערוויס-ענטפער. מען גיט style=service_style_pirate און text=service_reply. אזוי ווערן די פלאצהאלטערס פארביטן מיטן פיראט-סטיל און מיטן פאקטישן ענטפער. דער רעזולטאט ווערט געהאלטן אין service_messages. מען דרוקט service_messages[0].content, און עס ווייזט זיך דער פארמאטירטער מעסעדזש מיטן סערוויס-ענטפער.

צום סוף שיקט די שורה service_response = chat(service_messages) דעם פארמאטירטן מעסעדזש צו דער chat פונקציע, וואס רעדט מיטן OpenAI מאדעל. דער מאדעל באארבעט דעם מעסעדזש און שאפט א ענטפער לויט דעם פראמפט. דער ענטפער ווערט געהאלטן אין service_response. מען דרוקט דעם אינהאלט, און מען זעט דעם סערוויס-ענטפער איבערגעזעצט אינעם פיראט-טאן.

אויסגאנג-פארזערס: א JSON-ביישפיל (11:03:50)

אצינד הייבט זיך אן דער טעמע פון output parsers. מען זעט א JSON-אביעקט. JSON מיינט JavaScript Object Notation, און עס האלט דאטא אין פארעם פון שליסל און ווערט (key-value pairs). די טיילן זענען:

- gift: false: דער חפץ איז נישט קיין מתנה. false איז א boolean ווערט.
- delivery_days: 5: די צושטעל-צייט פונעם חפץ איז 5.
- price_value: "pretty affordable": א באשרייבונג פונעם פרייז, אז ער איז גאנץ ביליג.

דערנאך זעט מען צוויי וואריאבלען: customer_review און review_template. customer_review איז א סטרינג פון עטליכע שורות מיט א דעטאלירטער רעצענזיע אויף א) leaf blower בלעטער-בלאזער). די רעצענזיע ווערט אריינגעשטעלט אין דעם מוסטער דורך text. דער מאדעל דארף אויסגעבן די אינפארמאציע אין JSON-פארמאט, מיט די שליסלען gift, delivery_days און price_value. דער אויפבוי איז געמאכט כדי אויטאמאטיש ארויסצוציען און צו פארמאטירן די וויכטיגע פרטים פון דער רעצענזיע.

פונעם מוסטער ביז א פעלער (11:04:51)

ChatPromptTemplate.from_template(review_template) שאפט א פראמפט-מוסטער פון דעם review_template סטרינג. review_template באשטימט א סטרוקטורירטן פראמפט, וואס ווייזט דעם מאדעל ווי צו ציען באשטימטע אינפארמאציע פון א רעצענזיע. מיט from_template פארוואנדלט LangChain דעם סטרינג אין א ChatPromptTemplate אביעקט, און מען קען אים ניצן צו שאפן דינאמישע פראמפטס. מען דרוקט דעם מוסטער, כדי צו זען די סטרוקטור.

דערנאך פילט prompt_template.format_messages דעם מוסטער מיטן טעקסט פון דער רעצענזיע. מען שאפט אן ChatOpenAI אביעקט, ווואו model איז llm_model און temperature איז 0. די נול-טעמפעראטור מאכט די ענטפערס מער באשטימט און וויניגער צופעליג. דער מאדעל ענטפערט, און מען דרוקט די אינפארמאציע וואס ער האט ארויסגעצויגן אין JSON-פארמאט: צי דער חפץ איז געווען א מתנה, די צושטעל-צייט, און דער פרייז-ווערט.

אבער ווען מען קוקט אויפן טיפ פון response.content, זעט מען אז עס איז א פשוטער string. ווען מען פרובירט ניצן דעם רעזולטאט, קומט ארויס א פעלער: "'get' str' object has no attribute'". דער ענטפער איז נאר טעקסט, נישט קיין דיקשענערי.

די לייזונג: ResponseSchema און StructuredOutputParser (11:06:25)

כדי צו לייזן די פראבלעם, אימפארטירט מען צוויי קלאסן פון langchain.output_parsers: ResponseSchema און StructuredOutputParser. צוזאמען העלפן זיי סדר'ן און סטרוקטורירן די אינפארמאציע וואס דער מאדעל גיט ארויס, כדי מען זאל זי גרינגער קענען ניצן אין דער אפליקאציע.

מען שאפט דריי ResponseSchema אביעקטן, וואס באשטימען ווי דער אויסגאנג דארף אויסזען. יעדער שעמע שטעלט פאר איין פרט וואס דער מאדעל דארף ציען פון דער רעצענזיע: gift_schema, delivery_days_schema און price_value_schema. די שעמעס ווערן צוזאמענגעשטעלט אין א רשימה מיטן נאמען response_schemas. זי פירט דעם פארזינג, אז דער ענטפער זאל פאסן צום ערווארטעטן פארמאט פאר יעדן פרט.

די שורה StructuredOutputParser.from_response_schemas(response_schemas) שאפט א פארזער לויט די שעמעס. דער פארזער טייטשט דעם ענטפער פונעם מאדעל און זעט צו אז ער פאסט צו דער סטרוקטור. וועו מען גיט אים די רשימה, זאגט מען אים וועלכע דאטא צו ערווארטן און ווי זי צו ארגאניזירן. אזוי ווערט דער טעקסט פונעם מאדעל אויטאמאטיש פארוואנדלט אין א סטרוקטורירטן פארמאט, ווי א Python dictionary אדער JSON.

output_parser.get_format_instructions() ברענגט די פארמאט-אנווייזונגען. דאס זענען הויפט-אנווייזונגען פארן מוסטער, וואס זאגן דעם מאדעל ווי זיין ענטפער דארף אויסזען, כדי ער זאל פאסן צו די שעמעס. דאס מאכט לייכטער פארן פארזער שפעטער צו טייטשן די דאטא. מען דרוקט די אנווייזונגען, און זיי ווייזן א JSON-סטרוקטור.

דער סוף-רעזולטאט (11:08:58)

מען שרייבט א נייעם review_template מיט די אנווייזונגען, שאפט דעם מוסטער מיט ChatPromptTemplate.from_template, און פילט אים מיט prompt.format_messages. מען דרוקט דעם אינהאלט פונעם ערשטן מעסעדזש. דערנאך שיקט response = chat(messages) די מעסעדזשעס צו דער chat פונקציע, וואס רעדט מיטן OpenAI מאדעל, און דער ענטפער קען ווערן געוצט פאר ווייטערדיגער באארבעטונג אדער אויסווייזן. מען דרוקט response.content און זעט: gift איז true, delivery_days איז 2, און price_value איז: "It's slightly more expensive than the other leaf blowers out there but I think it's worth it for the extra features", דאס הייסט עס איז אביסל טייערער ווי אנדערע בלעטער-בלאזערס, אבער עס איז ווערט צוליב די עקסטערע מעגליכקייטן.

זכרון פאר מאדעלן: צוגרייטונג (11:09:28)

דער קומענדיגער קאפיטל באהאנדלט ווי מען גיט א מאדעל זכרון. אמאל איז עס זייער נויטיג אז א טשעטבאט זאל געדענקען דעם שמועס. למשל, ביי א קונה-סערוויס טשעט איז עס נוצליך אז די פריערדיגע שמועסן זאלן בלייבן געדענקט.

צוערשט שטעלט מען אן די סביבה פונעם פראיעקט. מען אימפארטירט דעם os מאדול, וואס העלפט ארבעטן מיט דער אפערירונג-סיסטעם. דערנאך ברענגט מען load_dotenv און find_dotenv פון דעם dotenv פעקל. זיי לאדן סביבה-וואריאבלען פון א .env טעקע, וואו מען האלט סענסיטיווע אינפארמאציע ווי API שליסלען. find_dotenv זוכט די טעקע, און load_dotenv לאדט איר אינהאלט אין דער סביבה, כדי די פראגראם זאל קענען צוקומען צו די וואריאבלען.

אויסקלייבן די מאדעל-ווערסיע לויטן דאטום (11:11:00)

דער קומענדיגער קאוד באהאנדלט דאס אז א געוויסע ווערסיע פונעם מאדעל ווערט אפגעשאפט (deprecated). ער אימפארטירט דעם datetime מאדול, וואס לאזט ארבעטן מיט דאטומס און צייטן אין Python. מיט datetime.datetime.now().date() נעמט מען דעם היינטיגן דאטום. דער ציל-דאטום איז דער 12טער יוני 2024. אויב דער היינטיגער דאטום איז שפעטער, שטעלט מען דעם מאדעל אויף gpt-3.5-turbo, מיטן חשבון אז נאך יענעם דאטום איז די עלטערע ווערסיע אפגעשאפט. אויב מען האלט נאך פאר דעם דאטום, ניצט מען די עלטערע ווערסיע gpt-3.5-turbo-0301.

LangChain קאמפאנענטן פארן זכרון (11:11:30)

צום סוף אימפארטירט מען קאמפאנענטן פון LangChain, וואס דער קורס באצייכנט אלס א שטארקן געצייג צו ארבעטן מיט שפראך-מאדעלן אויף א מער סטרוקטורירטן און אינטעראקטיוון אופן:

- ChatOpenAI פון langchain.chat_models : א קלאס וואס לאזט גרינג רעדן מיט די טשעט-מאדעלן פון OpenAI, ווי GPT-3 אדער GPT-4, כדי צו שאפן שמועס-ענטפערס.
- ConversationChain פון LangChain.
- ConversationBufferMemory פון langchain.memory: זי באהאנדלט דעם זכרון פונעם שמועס. זי האלט אן און ברענגט צוריק די פריערדיגע אויסטוישן, כדי דער שמועס זאל זיין מער צוזאמענהענגיק און נאטירלעך, ווייל דער מאדעל געדענקט וואס מען האט פריער געזאגט.

אלע די קאמפאנענטן ארבעטן צוזאמען כדי צו בויען אן אינטעראקטיווע טשעט-דערפארונג, וואו דער מאדעל האלט דעם קאנטעקסט און דעם פלוס פונעם גאנצן שמועס.

זכרון אין LangChain און די ערשטע טריט מיט chains (11:12:31)

א שמועס מיט ConversationBufferMemory (11:12:31)

אין דעם טייל פונעם קורס ווערט געבויט א שמועס-אפליקאציע (conversational AI) מיט דער LangChain ביבליאטעק. צום ערשט ווערט אויפגעשטעלט א וואריאבל LLM מיט דער קלאס ChatOpenAI. דער פאראמעטער temperature ווערט געשטעלט אויף 0.0. עס ווערט ערקלערט אז דאס מאכט די ענטפערס פונעם מאדעל "דעטערמיניסטיש": אויף דעם זעלבן אינפוט קומט יעדעס מאל דער זעלבער אויספיר. דאס איז נוצלעך ווען מען וויל האבן קאנטראל איבער די שמועסן. דער פאראמעטער model באקומט דעם וואריאבל llm_model, וואס איז פריער דעפינירט געווארן מיטן נאמען פונעם מאדעל.

דערנאך ווערט געשאפן א וואריאבל conversation מיט דער קלאס ConversationChain . דער ConversationChain בינדט אלעס צוזאם: ער פארבינדט דעם LLM מיט דעם זכרון (memory). ער שטעלט אויך verbose=True, כדי עס זאלן ארויסגעדרוקט ווערן פולע לאגס פונעם שמועס. אזוי איז גרינגער צו געפינען פעלערן (debug) אדער צו אנאליזירן וואס עס גייט פאר.

נאכדעם ווערט דער אינפוט "hi, my name is Andreo" געשיקט צום מאדעל. דער conversation אביעקט באארבעט דעם אינפוט און גיט א ענטפער לויטן קאנטעקסט וואס ליגט אין זכרון. דער מאדעל ענטפערט: "Hello, it's nice to meet you. How can I assist you today?". עס פירט זיך פונקט ווי א טשאט.

דערנאך ווערט געשיקט די פראגע "what is 1+1". דער ענטפער איז: "1+1 equals 2. Is there anything else you would like to know?".

דער עיקר טעסט קומט מיט conversation.predict(input="what is my name?"). דא בעט מען דעם מאדעל ער זאל זיך דערמאנען דעם נאמען. ווייל דער שמועס-זכרון איז אקטיוו, קוקט דער מאדעל צוריק אין די פריערדיגע שמועסן צו זען צי ער ווייסט שוין דעם נאמען. ער ענטפערט: "your name is Andreo". מען זעט קלאר אז דער מאדעל ארבעט מיט זכרון.

ארויסנעמען און אריינלייגן אינעם זכרון (11:14:14)

די שורה (memory.buffer)print ווייזט וואס עס ליגט אינעם שמועס-זכרון. דער memory איז אן אינסטאנץ פון ConversationBufferMemory, און ער היט אפ אלע אויסטוישן וואס זענען פארגעקומען אינעם שמועס. ווען מען דרוקט memory.buffer זעט מען די גאנצע היסטאריע פונעם שמועס.

דערנאך ווערט געגוצט (memory.load_memory_variables{}) צו לאדן די יעצטיגע זכרון-וואריאבלען. די גרעזל-קלאמערן זענען א ליידיגע dictionary. מען בעט דערמיט ארויסצונעמען דעם גאנצן זכרון ביז יעצט, אן קיין שום פילטער אדער באדינגונג. דאס העלפט צוצוקומען צו דער גאנצער שמועס-היסטאריע און צו פירן איר. מען באקומט א "בילד" פון אלעס וואס דער מאדעל געדענקט פון די שמועסן. מען זעט אז ער געדענקט טאקע אלעס.

דערנאך ווערט געשאפן א נייע אינסטאנץ פון ConversationBufferMemory. דאס איז א ספעציעלער סארט זכרון צו אפהיטן שמועס-היסטאריע. מיט memory.save_context ווערט אריינגעלייגט א נייער אויסטויש: "hi" input און "whats up" output". דער input "hi" " שטעלט פאר וואס דער באניצער האט געזאגט, און "whats up" output איז דעם מאדעלס ענטפער. save_context זאגט דעם ConversationBufferMemory ער זאל דאס אפהיטן ווי א פאר פון input און output, כדי דער מאדעל זאל שפעטער אינעם שמועס קענען צוריקקוקן דערויף.

דערנאך ווערט ווידער געדרוקט memory.buffer. ער ווייזט די גאנצע אפגעהיטענע היסטאריע, ווייל דער זכרון האלט חשבון פון יעדן באניצער-אינפוט און יעדן ענטפער. דאס איז נוצלעך פאר debug, אדער צו פארשטיין ווי אזוי דער מאדעל באארבעט דעם שמועס במשך דער צייט. memory.load_memory_variables מיט א ליידיגע dictionary איז, ווי פריער, א וועג ארויסצונעמען און דורכצוקוקן דעם אפגעהיטענעם זכרון אן פילטערס.

ConversationBufferWindowMemory – א "פענצטער" פון בלויז איין אויסטויש (11:16:44)

דא ווערט אימפארטירט ConversationBufferWindowMemory פון langchain.memory. מען שאפט אן אינסטאנץ דערפון מיט א פענצטער-גרייס פון איינס, k=1. דאס הייסט אז דער זכרון וועט האלטן בלויז דעם לעצטן אויסטויש, סיי דעם אינפוט סיי דעם אויספיר. די עלטערע היסטאריע ווערט אוועקגעווארפן. דער קורס ערקלערט אז אזא אויפשטעל איז גוט ווען מען וויל אז דער מאדעל זאל זיך קאנצענטרירן נאר אויפן גלייכן קאנטעקסט. דאס האלט די זאך לייכט און עפעקטיוו, בפרט ביי לאנגע שמועסן.

דערנאך ווערן אפגעהיטן צוויי פארן מיט (key-value) save_context:

- "whats up" output ,"hi" input
- "cool" output ,"not much, just hanging" input

נאכדעם ווערט גערופן load_memory_variables מיט א ליידיגע dictionary. דאס נעמט ארויס אלע זכרון-וואריאבלען וואס ליגן אינעם buffer, אן פילטערס.

דערנאך ווערט ווידער אויפגעשטעלט א שמועס. LLM איז ChatOpenAI מיט temperature אויף 0, כדי די ענטפערס זאלן זיין דעטערמיניסטיש. דער מאדעל ווערט אויסגעקליבן לויטן וואריאבל llm_model, וואס קען דינאמיש בייטן צווישן פארשידענע ווערסיעס. memory איז ConversationBufferWindowMemory(k=1), א זכרון וואס געדענקט נאר דעם לעצטן אויסטויש. conversation ווערט אויפגעשטעלט מיט

ConversationChain, וואס פארבינדט דעם מאדעל מיטן זכרון. דאס מאל איז verbose=False, אזוי אז די פולע לאגס ווערן נישט געוויזן און דער אויספיר בלייבט ריין.

מען שיקט "hi, my name is Andrew", און דער מאדעל ענטפערט נאטירלעך אויף דער פארשטעלונג פונעם באניצער. דאס איז דער אנהייב פונעם שמועס. דערנאך שיקט מען "what is 1+1". ווייל דאס איז א פשוטע חשבון-פראגע, ענטפערט דער מאדעל ריכטיג "2". מען זעט ווי דער שמועס בויט זיך דינאמיש פון יעדן אויסטויש. נאכדעם פרעגט מען נאכן נאמען, און דער מאדעל ענטפערט "your name is Andrew". די לעקציע ערקלערט אז ווייל דער זכרון איז אויפגעשטעלט צו האלטן פריערדיגע אויסטוישן, קען דער מאדעל צוריקקוקן אויפן אויסטויש וואו דער באניצער האט זיך פארגעשטעלט. אזוי שפירט זיך דער שמועס נאטירלעך און צוזאמענהענגיק.

ConversationTokenBufferMemory – זכרון לויט טאקענס (11:19:40)

דער קומענדיגער סארט זכרון איז ConversationTokenBufferMemory פון LangChain. ער היט אפ די שמועס-היסטאריע לויט דער צאל טאקענס. מען אימפארטירט אויך OpenAI פון langchain.llms. ווידער ווערט llm אויפגעשטעלט מיט ChatOpenAI, temperature 0.0, און דעם מאדעל לויט llm_model, כדי די אויספירן זאלן זיין קאנסיסטענט לויט דער געגעבענער ווערסיע.

דער זכרון ווערט געשאפן מיט max_token_limit=50. דאס הייסט אז די היסטאריע ווערט באגרעניצט צו 50 טאקענס אינגאנצן, כדי צו פירן דעם זכרון עפעקטיוו. מען היט אפ דריי אויסטוישן מיט save_context, יעדער מיט א באניצער-אינפוט און דעם מאדעלס ענטפער. אויב די סך-הכל צאל טאקענס גייט אריבער 50, קענען די נייסטע אויסטוישן ארויסשטופן די עלטערע.

דערנאך ווייזט memory.load_memory_variables({}) וואס עס ליגט יעצט אינעם זכרון, ווידער אן פילטערס. מען באקומט א בילד פון דער היסטאריע ביז יענעם פונקט, מיט די אינפוטן און אויספירן וואס דער מאדעל האט באארבעט. דאס איז נוצלעך צו איבערקוקן אדער צו ענדערן דעם זכרון במשך דעם שמועס.

וואס איז א chain אין LangChain (11:21:12)

נאכן זכרון גייט דער קורס איבער צו chains. א chain איז א סדר פון טריט אדער טיילן וואס ארבעטן צוזאמען צו אויספירן א געוויסע ארבעט. די טריט קענען זיין באזונדערע אפעראציעס, למשל שאפן טעקסט מיט א שפראך-מאדעל אדער אפהיטן דאטא אין זכרון. א chain פארבינדט די פארשידענע טיילן פונעם פראצעס: דעם מאדעל, דעם זכרון און די געצייג (tools). אזוי פליסט אלעס גלאט פון איין טריט צום צווייטן. דער משל: א שטאפעט-לויף (relay race), וואו יעדער לויפער גיט איבער דעם שטעקן צום קומענדיגן, און אזוי ענדיגט מען די ארבעט ווי א מאנשאפט.

פארוואס chains זענען וויכטיג:

- זיי ארגאניזירן ארבעטן. זיי געבן סטרוקטור און מאכן גלאטער ווי אזוי פארשידענע ארבעטן ווערן אויסגעפירט.
- אן chains ווערט עס קאמפליצירט. מען דארף באהאנדלען יעדע אפעראציע באזונדער, בפרט ווען מען פארבינדט פארשידענע מאדעלן, געצייג אדער מקורות פון אינפארמאציע.
- זיי שאפן א קלארן סדר פון ארבעט (workflow). דאס איז א פראצעס וואס מען קען איבערחזרן, וואו יעדער טריט בויט זיך אויפן פריערדיגן.

• זיי העלפן וואקסן מיט קאמפליצירטע ארבעטן. מען קען גרינגער פירן און פארגרעסערן קאמפליצירטע סיסטעמען, אזוי אז אלע טיילן ארבעטן צוזאמען.

• זיי פארפשוטן און אויטאמאטיזירן. זיי פארמינערן האנט-ארבעט, באהאנדלען איבערגעחזרטע ארבעטן אויטאמאטיש, און שפארן צייט און מי.

ווי אזוי א chain ארבעט:

• מען גיט א אינפוט צום chain.

• דער אינפוט גייט צום ערשטן טריט. דאס קען זיין א שפראך-מאדעל אדער א געציג וואס שאפט א ענטפער לויטן אינפוט.

• דער ענטפער גייט ווייטער צום קומענדיגן טריט. דארט ווערט ער אפגעהיטן אין זכרון, געענדערט, אדער מען טוט אן אנדער אקציע, ווי באארבעטן דאטא.

• דער לעצטער טריט באארבעט אלעס און גיט ארויס דעם רעזולטאט: א ענטפער, אן אקציע, אדער נאך א תגובה.

• הינטער די קוליסן באהאנדלט דער chain אלע טריט אויטאמאטיש. מען דארף נישט זארגן פאר יעדער באזונדערער אפעראציע, און מען קען זיך קאנצענטרירן אויף דער העכערער ארבעט.

א צווייטער משל: א פאבריק-ליניע (assembly line). די רויע מאטעריאלן, דאס הייסט די אינפוטן, ווערן באארבעט אין פארשידענע סטאנציעס, ביז עס קומט ארויס דער ענדגילטיגער פראדוקט, דאס הייסט דער אויספיר.

צוגרייטן דעם קאוד פאר chains (11:24:19)

צום ערשט נוצט דער קאוד Python'ס warnings מאדול. דאס איז אן איינגעבויטע סיסטעם וואס ווארנט ווען עפעס קען גיין שלעכט אינעם קאוד. דא ווערן די ווארענונגען פארשטילט. דאס איז נוצלעך ווען דער קאוד גיט אומשעדלעכע ווארענונגען און מען וויל א ריינערן אויספיר. דער קורס ווארנט אבער מען זאל אכטונג געבן נישט צו פארפעלן עפעס וויכטיגס.

דערנאך ווערט גענוצט dotenv צו לאדן environment variables פון א .env טעקע. מען אימפארטירט load_dotenv צו לאדן די ווריאבלען און find_dotenv צו געפינען די .env טעקע:

```
load_dotenv(find_dotenv()) = _
```

דער "_" איז א געווענלעכער אופן צו זאגן: "איך דארף נישט דעם רעזולטאט, טו עס נאר". אזוי ליינט די פראגראם זיכער די פריוואטע איינשטעלונגען פון א קליינער טעקע, אן זיי שרייבן גלייך אינעם קאוד.

דערנאך קומט דער קאוד וואס איז שוין פריער ערקלערט געווארן: ער טשעקט צי דער היינטיגער דאטום איז שוין נאך א געוויסן דאטום, דעם 12טן יוני 2024, און לויט דעם באשליסט ער וועלכע ווערסיע פונעם שפראך-מאדעל צו נוצן.

דערנאך ווערט אינסטאלירט pandas. מען אימפארטירט pandas as pd און ליינט א CSV טעקע. מיטן ווייזן די ערשטע עטלעכע שורות פון דער טעקע זעט מען פינף שורות, נומערירט 0 ביז 4.

ChatOpenAI מיט העכערער temperature און א פראמפט-מוסטער (11:25:54)

מען אימפארטירט פון LangChain: ChatOpenAI, ChatPromptTemplate און LLMChain. מען שאפט א נייע אינסטאנץ פון ChatOpenAI, דאס מאל מיט temperature 0.9. דער קורס ערקלערט אז temperature קאנטראלירט ווי שעפעריש אדער צופעליג די ענטפערס זענען. א הויכע temperature ווי 0.9 מאכט די ענטפערס מער פארשידנארטיג און ווייניגער פאראויסזאגבאר. א נידעריגע temperature מאכט זיי מער פאקוסירט און דעטערמיניסטיש.

דערנאך ווערט געשאפן א פראמפט-מוסטער מיט ChatPromptTemplate.from_template. דער מוסטער פרעגט: וואס איז דער בעסטער נאמען פאר א פירמע וואס מאכט {product}. {product} איז א פלאץ-האלטער, וואס ווערט אויסגעפילט מיט א פאקטישן פראדוקט-נאמען ביים נוצן. מען קען אריינלייגן יעדן פראדוקט, למשל בייסיקלען אדער גאדזשעטס, און דער מאדעל ענטפערט מיט א פארגעשלאגענעם פירמע-נאמען לויט יענעם פראדוקט.

קייטן אין LangChain: פון איינפאכע ביז ראוטינג (11:26:56)

א LLMChain מיט א פראדוקט-נאמען (11:26:56)

דער קורס ערקלערט אז מען וויל מאכן דעם פראמפט בייגעוודיג און ווידער-באנוצבאר פאר פארשידענע מצבים. דערפאר שאפט מען א נייעם LLMChain אביעקט, וואס איז א וועג צו פארבינדן דעם גרויסן שפראך-מאדעל מיט א געווישן פראמפט.

אין דעם קאוד באקומט דער וועריעבל product דעם ווערט "Queen Size Sheet Set". דאס איז דער פראדוקט פאר וועלכן דער מאדעל וועט שאפן אן ענטפער. די שורה chain.run(product) לאזט לויפן דעם מאדעל דורכן llm אביעקט מיטן געגעבענעם פראמפט. דער ארט-האלטער (placeholder) ווערט אויסגעפילט מיט "Queen Size Sheet Set". אזוי באקומט דער מאדעל דעם פראמפט "וואס איז דער בעסטער נאמען צו באשרייבן א פירמע וואס מאכט א Queen Size Sheet Set?", און ער שאפט אן ענטפער דערויף.

SimpleSequentialChain: צוויי קייטן איינס נאכן אנדערן (11:27:28)

דערנאך אימפארטירט מען SimpleSequentialChain פונעם מאדול langchain.chains. עס ווערט ערקלערט אז דאס איז א געצייג אין LangChain וואס לאזט שאפן א סדר פון אויפגאבעס. דער אויסגאנג פון יעדער אויפגאבע ווערט דער איינגאנג פאר דער נעקסטער. אט אזוי בויט מען אן:

- llm = ChatOpenAI(...) שאפט אן אינסטאנץ פונעם ChatOpenAI מאדעל מיט א שעפעריישקייט-שטאפל (temperature) פון 0.9, און גיט אן וועלכן מאדעל מען זאל באניצן.
- first_prompt = ChatPromptTemplate... שאפט א פראמפט-מוסטער וואס פרעגט דעם מאדעל וואס איז דער בעסטער פירמע-נאמען פאר א פראדוקט. דער טייל product איז א ארט-האלטער, וואס ווערט שפעטער פארביטן מיטן אמת'ן פראדוקט-נאמען.
- chain_one = LLMChain(...) פארבינדט דעם מאדעל מיטן פראמפט. די קייט נעמט דעם פראדוקט ווי איינגאנג, לייגט עס אריין אינעם מוסטער, און שאפט אן ענטפער מיט א פארשלאג פאר א פירמע-נאמען.
- דערנאך שאפט מען א צווייטן פראמפט-מוסטער, second_prompt, וואס בעט דעם מאדעל צו שרייבן א באשרייבונג פון 20 ווערטער פאר א פירמע. מען שאפט אויך chain_two, וואס פארבינדט דעם זעלבן

מאדעל מיט דעם נייעם פראמפט. דער געדאנק: נאכדעם וואס די ערשטע קייט שאפט א פירמע-נאמען, נעמט די צווייטע קייט דעם נאמען און שאפט א קורצע באשרייבונג פון 20 ווערטער.

• דערנאך שאפט מען overall_simple_chain, א SimpleSequentialChain וואס שטעלט צוזאמען די צוויי קייטן שריט נאך שריט. דער באשטאנד verbose=True זארגט או דער פראצעס זאל אויסדרוקן פרטים איבער יעדן שריט ווען ער לויפט. אזוי זעט מען וואס עס טוט זיך בעת דער דורכפירונג.

• ווען מען רופט overall_simple_chain.run(product), ווערט דער גאנצער פראצעס אויסגעלייזט מיט איין מאל. דער מאדעל גיט ארויס סיי דעם פירמע-נאמען און סיי די באשרייבונג, באזירט אויפן פראדוקט וואס מען האט אריינגעגעבן.

SequentialChain: פיר קייטן מיט באנאמענטע אויסגאנגען (11:30:04)

דערנאך הייבט מען אן מיט SequentialChain, וואס מען אימפארטירט פון LangChain. אט זענען די פיר שריט:

• prompt_template_one: "איבערזעצן די פאלגנדע רעצענזיע (review) אויף ענגליש". chain_one האט "English_Review"output_key=".

• אויפן זעלבן אופן גייט second_prompt : "קענסטו צוזאמפאסן די פאלגנדע רעצענזיע אין איין זאץ?". chain_two האט "summary"output_key=".

• third_prompt: "אין וועלכע שפראך איז די פאלגנדע רעצענזיע?". chain_three האט "language"output_key=".

• עס איז דא אויך א פערטער מוסטער מיט chain_four.

דערנאך שאפט מען overall_chain , וואס שטעלט צוזאמען אלע פיר קייטן אין איין המשך'דיגן פראצעס דורך SequentialChain. די קייט נעמט א רעצענזיע ווי איינגאנג. קודם זעצט זי איבער די רעצענזיע, דערנאך פאסט זי זי צוזאם, דערנאך דערקענט זי די שפראך, און צום סוף שאפט זי א נאכפאלג-מעסעדזש. דער באשטאנד input_variables=["Review"] זאגט אז דער איינגאנג פון דער גאנצער קייט איז די רעצענזיע. output_variables זאגן וועלכע ווערטן קומען ארויס. verbose=True זארגט או די פרטים פון יעדן שריט ווערן אויסגעדרוקט, כדי עס זאל זיין קלאר.

דערנאך, אין df.Review[5] = review, קלייבט מען אויס די זעקסטע רעצענזיע פונעם קאלאם Review אינעם data frame מיטן נאמען df. עס איז די זעקסטע, ווייל די נומערירונג הייבט זיך אן פון נול. מען היט עס אפ אינעם וועריעבל review.

פיר מוסטערן לויט פאך-געביטן (11:32:06)

דער קורס ווייזט פיר פארשידענע מוסטערן, וואס פירן די ענטפערס פונעם שפראך-מאדעל לויטן טעמע פון דער פראגע. יעדער מוסטער שטעלט אוועק דעם קאנטעקסט און דעם טאן, ווי אזוי דער מאדעל זאל ענטפערן לויטן פאך:

• physics template: דער מאדעל איז א קלוגער פראפעסאר פון פיזיק, וואס ענטפערט פראגעס אויף א קלארן און קורצן אופן.

• math template: דער מאדעל איז א גוטער מאטעמאטיקער, וואס ענטפערט דורכן צעטיילן קאמפליצירטע פראבלעמען אין פשוט'ערע טיילן.

- history template: דער מאדעל איז אן עקספערט היסטאריקער, מיט טיפע ידיעות איבער מענטשן, געשעענישן און קאנטעקסטן פון פארשידענע היסטארישע תקופות.
- computer science template: דער מאדעל איז א געלונגענער קאמפיוטער-וויסנשאפטלער, מיט שטארקע כוחות צו לייזן פראבלעמען, ידיעות איבער אלגאריטמען, און די פעאיקייט צו ערקלערן קאוד-לייזונגען קלאר.

יעדער מוסטער האט א ארט-האלטער input, וואו מען לייגט אריין די אמת'ע פראגע.

דערנאך שאפט מען א רשימה prompt_infos, וואס היט אפ אינפארמאציע איבער די פיר געביטן: פיזיק, מאטעמאטיק, היסטאריע און קאמפיוטער-וויסנשאפט. יעדער איינס איז א dictionary מיט דריי שליסלען: name, description און prompt_infos. prompt_template איז אזוי א זאמלונג פון מוסטערן און באשרייבונגען פאר יעדן געביט. מען קען זי באניצן צו אויסקלייבן דעם ריכטיגן מוסטער, וועו מען פרעגט א פראגע איבער א געוויסן טעמע.

דער ראוטער: MultiPromptChain (11:33:41)

מען אימפארטירט MultiPromptChain, LLMRouterChain, RouterOutputParser און PromptTemplate פון langchain.chains און פון prompts. דערנאך שאפט מען אן אינסטאנץ פון ChatOpenAI מיט temperature=0. דאס מיינט אז די ענטפערס וועלן זיין זייער פאקוסירט און דעטערמיניסטיש. דער טייל model=llm_model זאגט וועלכע וועדסיע פונעם מאדעל מען באניצט; llm_model איז א וועריעבל וואס האלט דעם מאדעל-נאמען, למשל GPT-3.5 Turbo. דאס איז דער שריט פון איינשטעלן.

דערנאך שטעלט דער קאוד אויף א דינאמישע סיסטעם: עס ווערן געשאפן עטליכע קייטן לויט די געביטן, און זיי ווערן אפגעהיטן אין א dictionary. צום סוף ווערט די רשימה destinations צוזאמגעשטעלט אין איין סטרינג, destinations_str, וואו יעדער איינס שטייט אויף א באזונדערער שורה. דער סטרינג אנטהאלט א שיין צוגעשטעלטע רשימה פון אלע געביטן מיט זייערע באשרייבונגען. דאס קען זיין נוצליך כדי צו ווייזן אפציעס אדער אנווייזונגען פארן באניצער.

דערנאך שאפט מען א default_prompt דורך ChatPromptTemplate.from_template. דאס איז א פשוט'ער מוסטער וואס נעמט וועלכן איינגאנג עס זאל נאר זיין, און גיט אים איבער צום מאדעל אן קיין באזונדערע פארמאטירונג אדער ענדערונג. דער ארט-האלטער input זארגט אז דער טעקסט ווערט באניצט גלייך ווי ער איז. אזוי ווערט דער מוסטער א "כאפ-אלעס" פאר יעדן איינגאנג.

דערנאך קומט MULTI_PROMPT_ROUTER_TEMPLATE, וואס איז געמאכט צו אויסקלייבן דעם פראמפט וואס פאסט בעסטנס פאר א געגעבענעם איינגאנג. דער מאדעל באקומט עטליכע פראמפטן מיט באשרייבונגען, און ער דארף אויסקלייבן איינעם וואס פאסט צום איינגאנג. דער מאדעל קען אויך ענדערן דעם איינגאנג, אויב דאס וועט ברענגען א בעסערן ענטפער. דער אויסגאנג איז א markdown קאוד-שטיקל מיט א JSON אביעקט, וואס אנטהאלט צוויי וועדסן: destination און next_inputs. דער אפטייל candidate prompts רעכנט אויס די פאראנענע פראמפטן, און דער ארט-האלטער input האלט דעם רויען טעקסט. אזוי פירט דער מוסטער דעם איינגאנג צום פראמפט וואס איז שייך צו זיין אינהאלט.

דערנאך פארמאטירט מען dעם router template מיט destinations_str, און דאס שטעלט אויף א קלוגע ראוטינג-סיסטעם. זי שאצט אפ דעם איינגאנג און פירט אים צום פאסיגן פראמפט, אזוי אז די סיסטעם איז

בייגעוודיג פאר אלערליי פראגעס. צום סוף שטעלט מען אויף א MultiPromptChain, וואס פאראייניקט די ראוטינג-סיסטעם מיט די עטליכע destination chains אין איין אינאיינעם ארבעטס-גאנג. עס איז דא אויך א רעזערוו-אפציע פאר אלגעמיינע פראגעס.

צוויי פראבעס: פיזיק און אלגעמיין (11:37:16)

- chain.run("What is black body radiation?"): קודם שאצט די router chain אפ די פראגע. ווייל די פראגע באציט זיך צו א פיזיק-באגריף, וועט זי מסתמא פירן דעם איינגאנג צום physics פראמפט. דערנאך נעמט די סיסטעם די פאסיגע פיזיק-קייט פונעם dictionary destination_chains, כדי צו באארבעטן די פראגע. ווייל verbose=True איז אנגעשטעלט, דרוקט זי אויס פרטים איבערן ראוטינג און איבערן ענטפער. אזוי זעט מען קלאר ווי אזוי דער איינגאנג איז באארבעט געווארן, און וועלכער פראמפט איז באניצט געווארן.
- chain.run("Why does every cell in our body contain DNA?"): די router chain שאצט אפ די פראגע, און די default chain באהאנדלט זי. דער מאדעל שאפט אן אלגעמיינעם ענטפער, וואס קען ערקלערן די וויכטיקייט פון DNA אין די צעלן. מיט verbose=True ווערן די ראוטינג-באשלוסן אויסגעשריבן: וועלכער פראמפט איז אויסגעקליבן געווארן, און וואס פארא ענטפער איז ארויסגעקומען.

קלוגע פראגע-ענטפער (QA) סיסטעמען מיט LangChain (11:38:49)

דער נעקסטער קאפיטל באהאנדלט דאס בויען קלוגע QA סיסטעמען מיט LangChain. QA, אדער Question Answering) פראגע-ענטפער), איז אן אויפגאבע אינעם געביט פון natural language processing. דער ציל איז צו אנטוויקלען מאדעלן אדער סיסטעמען וואס פארשטייען און ענטפערן פראגעס וואס ווערן געשטעלט אין א נאטירלעכער שפראך. בעצם לערנט מען מאשינען צו ענטפערן פראגעס באזירט אויף א געגעבענעם טעקסט-אפשניט, אויף א וויסן-באזע, אדער אפילו אויף זייער פארשטאנד פון דער וועלט.

פארוואס איז QA אזוי שטארק? ווייל עס לאזט מאשינען זיך אפגעבן מיט מענטשן אויף אן אופן וואס איז ענלעך צו ווי מיר באארבעטן און ענטפערן אויף אינפארמאציע:

- גלייכצייטיגע ענטפערס: QA סיסטעמען ליפערן שנעל געניע ענטפערס אויף א ברייטע ריי פראגעס. דאס איז זייער נוצליך אין געביטן ווי קונה-סערוויס, חינוך אדער פארשונג.
- א בעסערע באניצער-דערפארונג: מיט געניע, מענטש-ענלעכע ענטפערס באקומען מענטשן די אינפארמאציע וואס זיי דארפן מער עפעקטיוו.
- באהאנדלען קאמפליצירטע פראגעס: פארגעשריטענע QA סיסטעמען, באזונדערס ווען זיי זענען פארבונדן מיט LLMs, קענען באארבעטן קאמפליצירטע פראגעס מיט עטליכע שריט. אזעלכע פראגעס פאדערן א שכל'דיגן חשבון אדער פארשטאנד, מער ווי בלויז געדענקען פאקטן.

ווי ארבעט QA?

- דער באניצער גיט א פראגע, פון א פשוט'ער ביז א קאמפליצירטער.
- אין LangChain ווערט די פראגע געפירט דורך פראמפט-מוסטערן אדער קייטן, כדי זי צו פארפיינערן פאר דער באארבעטונג.
- דער שפראך-מאדעל באארבעט דעם איינגאנג און שאפט אן ענטפער באזירט אויף זיינע ידיעות.
- דער אויסגאנג ווערט געוויזן צום באניצער.

- אויב נויטיג, קען א feedback-שלייף בעטן אן ערקלערונג ביי אומקלארע אדער קאמפליצירטע פראגעס.

פראגע-ענטפער סיסטעם, אפשאצונג, און אגענטן מיט LangChain (11:40:49)

צוגרייטן די סביבה און לאדן דעם CSV-טעקע (11:40:49)

דער קורס הייבט אן מיטן אינסטאלירן אן אפגעדיטעטע ווערסיע פון LangChain, און מען לויפט דעם באפעל. דערנאך ווערט די Python-סביבה צוגעגרייט אזוי אז מען זאל קענען זיכער היטן סענסיטיווע אינפארמאציע, ווי API-שליסלען אדער פאסווערטער צו א דאטא-באזע.

דערצו ווערן אימפארטירט די פונקציעס load_dotenv און find_dotenv פונעם dotenv פעקל. עס ווערט ערקלערט ווי זיי ארבעטן:

- find_dotenv זוכט א טעקע מיטן נאמען .env. עס הייבט אן זוכן פונעם יעצטיגן פאלדער און גיט צוריק דעם פאד צו דער ערשטער .env טעקע וואס עס געפינט.
- load_dotenv לאדט די "environment variables" פון יענער טעקע אריין אין דער סביבה.

דערנאך קומט א שטיקל קאוד וואס קלייבט אויס די ריכטיגע ווערסיע פונעם מאדעל GPT-3.5 Turbo לויטן היינטיגן דאטום, ווייל געוויסע ווערסיעס ווערן מיט דער צייט אפגעשטעלט. דער קאוד נעמט דעם היינטיגן דאטום און שטעלט א ציל-דאטום פון 12טן יוני 2024. דער דאטום איז וויכטיג ווייל ארום יענער צייט האט מען אפגעשטעלט געוויסע ווערסיעס, למשל GPT-3.5 Turbo 0613. אין דער לעקציע ווערט באמערקט אז דאס איז שוין פריער ערקלערט געווארן.

דערנאך ווערן אימפארטירט: RetrievalQA, ChatOpenAI, CSVLoader, DocArrayInMemorySearch, display און Markdown, און OpenAI. זיי קומען פון די LangChain מאדולן chains, chat_models, document_loaders און vectorstores, און פון IPython.

מיטן CSVLoader לאדט מען דאטא פון א CSV-טעקע וועגן "outdoor clothing" (קלייד־ער פאר דרויסן). מען שאפט אן אינסטאנץ פון CSVLoader מיטן פאד צו דער טעקע. דער לאדער לייענט דעמאלט די טעקע און מאכט פון יעדער שורה א באזונדערן document אביעקט.

א וועקטאר-אינדעקס און די ערשטע פראגע (11:42:54)

מען אימפארטירט VectorStoreIndexCreator פון langchain.indexes , און מען אינסטאלירט דאס פעקל docarray. דערנאך בויט מען א אינדעקס כדי שנעל צו קענען זוכן און ארויסנעמען דאקומענטן. דאס טוט מען מיט VectorStoreIndexCreator צוזאמען מיט DocArrayInMemorySearch:

vectorstore_cls=DocArrayInMemorySearch

מיט דער באשטימונג קלייבט מען אויס א וועקטאר-שטאר וואס ליגט אינעם זכרון (in-memory) און וואס ערלויבט פונקטליכע זוכונגען.

מען שטעלט א פראגע וועגן העמדער וואס שיצן פון דער זון. דער ענטפער וואס מען באקומט זאגט אז ביים אויסקלייבן אזעלכע העמדער דארף מען קוקן אויפן מאטעריאל, ווי געדיכט דער שטאף איז געווענבט, און אויף דער קאליר. די אלע זאכן באווירקן ווי גוט דאס העמד שיצט פון UV-שטראלן. דער ענטפער דערמאנט אויך

פירמעס ווי Solumbra, וואס מאכן קליידער מיט א הויכן UPF-ראנג, וואס בלאקירן א גרויסן טייל פון די UV-שטראלן.

עס ווערט ערקלערט אז מען בויט דא א פראגע-ענטפער סיסטעם מיטן מאדעל GPT-3.5 Turbo Instruct פון OpenAI. מען באשטימט אז דער מאדעל איז OpenAI, אז temperature איז 0, און אז דער מאדעל איז GPT-3.5 Turbo. אזוי ווייסט די סיסטעם מיט ווען מאדעל צו שאפן די ענטפערס. (אין דער לעקציע ווערט דא דערמאנט LlamaIndex, אבער דער קאוד איז אין .LangChain)

דערנאך ניצט מען display און Markdown פון IPython כדי צו ווייזן דעם ענטפער שיין פארמאטירט אינעם Jupyter notebook. אז מען גיט דעם ענטפער אריין אין Markdown, ווייזט IPython דעם אינהאלט מיט דער ריכטיגער Markdown-פארמאטירונג, און מען זעט דעם רעזולטאט.

דער פראצעס שריט ביי שריט: דאקומענטן און embeddings (11:44:27)

יעצט ווייזט דער קורס דעם זעלבן פראצעס פונקט שריט נאך שריט. ווידער לאדט מען די CSV -טעקע מיטן CSVLoader. יעדע שורה ווערט א document, און יעדע קאלאָנע אין דער שורה ווערט א פארל פון שליסל און ווערט (key-value) אינעם אינהאלט פונעם דאקומענט.

די מעטאד loader.load() לייענט די טעקע און שאפט פון יעדער שורה א document אביעקט. מען קוקט אויפן ערשטן דאקומענט און מען זעט דעם רעזולטאט.

דערנאך שאפט מען אן אינסטאנץ פון דער קלאס OpenAIEmbeddings פון LangChain. די קלאס מאכט פון טעקסט וועקטארן מיט די מאדעלן פון OpenAI. עס ווערט ערקלערט אז embeddings זענען וועקטארן פון נומערן וואס כאפן דעם באדייט פונעם טעקסט. דאס מאכט מעגליך אפליקאציעס ווי זוכן לויטן באדייט (semantic search), גרופירן ענליכע זאכן (clustering), און רעקאמענדאציע-סיסטעמען. וועו מען שאפט די embeddings אזוי, ניצט די סיסטעם דעם גרונט-מאדעל (default) פון OpenAI פאר embeddings.

מיט דער מעטאד embed_query פארוואנדלט מען דעם טעקסט "Hi my name is Harrison" אין א וועקטאר פון נומערן. דאס כאפט דעם באדייט פונעם טעקסט, אזוי אז מען קען אים נאכדעם זוכן און אנאליזירן.

דערנאך קוקט מען ווי לאנג דער וועקטאר איז. דער מאדעל text-embedding-ada שאפט embeddings מיט 1536 דימענסיעס, דאס הייסט אז דער וועקטאר איז 1536 נומערן לאנג.

מיט print(embed[:5]) ווייזט מען די ערשטע פינף נומערן פונעם וועקטאר פאר דעם זאץ "Hi my name is Harrison". עס ווערט ערקלערט אז אין Python נעמט די נאטאציע [:5] די עלעמענטן פונעם אנהייב ביז אינדעקס 5, אבער אינדעקס 5 אליין איז נישט אריינגערעכנט. אזוי קען מען זען די ערשטע טיילן פונעם embedding.

זוכן ענליכע דאקומענטן און ענטפערן (11:47:00)

מיט דער שורה

db = DocArrayInMemorySearch.from_documents(...)

שאפט מען א וועקטאר-שטאר אינעם זכרון, וואס ערלויבט שנעל צו זוכן ענליכע זאכן אין דעם דאטא-סעט. מען שטעלט די פראגע "Please suggest a shirt with sunblocking" (שלאג פאר א העמד וואס בלאקירט די זון) און מען רופט similarity_search. מען באקומט צוריק פיר דאקומענטן, און מען קוקט אויפן ערשטן פון זיי.

דערנאך מאכט מען פון db א retriever, און מען רופט ווידער ChatOpenAI מיט temperature=0 און מיטן אויסגעקליבענעם מאדעל. מען ניצט א for-לופ איבער אלע דאקומענטן (for i in range(len(docs))), און מען גיט זיי איבער צום llm כדי צו באקומען א ענטפער. דאס ווייזט מען מיט display.

דערנאך בויט מען RetrievalQA מיט chain_type="stuff", llm=llm, און verbose=True, און מען גיט אריין א פראגע. מען זעט דעם ענטפער, און מען ווייזט אים ווי Markdown. צום סוף ווייזט מען אויך די קורצע מעטאד: index.query(query, llm=llm). עס ווערט אויך געוויזן אז מען קען אליין שאפן embeddings און א וועקטאר-שטאר, וועו מען רופט דעם loader.

ווי אזוי מען שאצט אפ די סיסטעם (11:48:37)

נאכדעם וואס מען האט א גרונטיגן באגריף ווי פראגע-ענטפער ארבעט, גייט דער קורס איבער צו דער פראגע ווי מען שאצט אפ די סיסטעם וואס מען האט געבויט. ווידער אימפארטירט מען os, וואס לאזט ארבעטן מיטן אפעריונג-סיסטעם און מיט די environment variables, און מען אימפארטירט load_dotenv און find_dotenv. אין דער לעקציע ווערט געזאגט אז דאס איז שוין ערקלערט געווארן צוויי-דריי מאל, און מען ערקלערט עס נישט נאכאמאל. ווידער אימפארטירט מען RetrievalQA, ChatOpenAI, CSVLoader, VectorStoreIndexCreator און DocArrayInMemorySearch.

דער קאוד לאדט די טעקע OutdoorClothingCatalog_1000.csv. ערשט שטעלט מען דעם פאד אריין אין דער וועריעבל file, און דערנאך לאדט מען די טעקע מיט CSVLoader פון langchain.document_loaders.

דערנאך שאפט מען א אינדעקס מיט VectorStoreIndexCreator. מען באשטימט DocArrayInMemorySearch ווי די קלאס פאר דעם וועקטאר-שטאר, וואס האלט די embeddings פון די דאקומענטן אינעם זכרון כדי שנעל צו קענען זוכן. די מעטאד from_loaders לאדט די דאטא פונעם loader וואס מען האט פריער באשטימט, באארבעט זי, און שאפט א אינדעקס אינעם זכרון. מיט דעם אינדעקס קען מען שנעל ארויסנעמען דאקומענטן לויט די embeddings.

דערנאך בויט מען די פראגע-ענטפער סיסטעם מיט RetrievalQA:

- מען שאפט דעם שפראך-מאדעל מיט ChatOpenAI, און מען שטעלט temperature=0.0, כדי די ענטפערס זאלן זיין פעסט און זיך נישט טוישן פון איין מאל צום צווייטן (deterministic).
- מען בויט די סיסטעם מיט from_chain_type, מיטן "stuff" שטייגער, און מען ניצט דעם וועקטאר-שטאר ווי דעם retriever, וואס ברענגט די פאסיגע דאקומענטן פאר יעדער פראגע.
- מען שאלט אריין verbose, כדי מען זאל קענען נאכפאלגן דעם פראצעס, און מען באשטימט אן אייגענעם טיילער צווישן די דאקומענטן (document separator).

פראגעס פאר די אפשאצונג: מיט דער האנט און אויטאמאטיש (11:50:41)

יעצט דארף מען אויסטראכטן פראגעס מיט ריכטיגע ענטפערס, כדי צו קענען אפשאצן די סיסטעם. data[10] נעמט דעם 11טן עלעמענט, און data[11] נעמט דעם 12טן, ווייל מען הייבט אן ציילן פון נול. אויב data אנטהאלט די שורות פון דער קאטאלאג-טעקע מיט 1000 שורות, גיט עס די 12טע שורה. ווי די דאטא קוקט אויס הענגט אפ ווי מען האט געלאדנט די CSV. למשל, אויב data איז א ליסטע פון dictionaries, גיט data[11] דעם dictionary וואס שטימט מיט דער 12טער שורה.

דערנאך שרייבט מען א ליסטע מיטן נאמען examples מיט צוויי dictionaries. יעדער פון זיי איז א פראגע מיט אן ענטפער: א query מיט דער פראגע, און אן answer מיטן ריכטיגן ענטפער. די צוויי פראגעס זענען: "צי האט דער Cozy Comfort Pullover Set זייט-קעשענעס?" און "פון וועלכער קאלעקציע איז דער Ultra-Lofty 850 Stretch Down Hooded Jacket?" אזא סטרוקטור קען מען ניצן כדי צו טריינירן אדער צו טעסטן א פראגע-ענטפער סיסטעם.

כדי נישט אלעס צו דארפן שרייבן מיט דער האנט, אימפארטירט מען QAGenerateChain פון langchain.evaluation.qa. מען שאפט א example_gen_chain מיט QAGenerateChain.from_llm, און מען גיט אים ChatOpenAI מיטן אויסגעקליבענעם מאדעל. די קייט שאפט אליין פארלעך פון פראגעס און ענטפערס.

דערנאך וועגדט מען אָן די קייט אויף די ערשטע פינף דאקומענטן אין data. מען ניצט א list comprehension וואס שאפט א ליסטע פון dictionaries, וועו יעדער dictionary האלט א דאקומענט אונטערן שליסל "doc". מען קוקט אויפן ערשטן ארויסגעקומענעם בייַשפּיל, און מען פארגלייכט עס מיט דער ערשטער דאטא (dem page_content פונעם דאקומענט). דערנאך לייגט מען די נייע בייַשפּילן צו צו דער ליסטע examples.

Debug מאָדע, און דער אנהייב פון אגענטן (11:53:46)

מען לויפט דעם ערשטן query פון דער ליסטע examples, דאס הייסט די פראגע וועגן די זייט-קעשענעס פונעם Cozy Comfort Pullover Set.

עס ווערט ערקלערט אז וועו מען שטעלט langchain.debug = True, הייסט מען LangChain ווייזן געוויע אינפארמאציע וואס עס טוט הינטער די קוליסן. עס דרוקט אויס debug-מעלדונגען, טעות-מעלדונגען, און די שריט וואס עס מאכט בשעת דער קאוד לויפט. מען לויפט ווידער דעם ערשטן query און מען זעט דעם debug-ארויסגאנג. דערנאך שאלט מען אויס דעם debug מאָדע.

דערנאך הייבט זיך אן א נייע לעקציע: ווי אזוי בויט מען אגענטן מיט LangChain . ערשט אימפארטירט מען os, און מען אימפארטירט load_dotenv פון dotenv כדי צו לייענען די לאקאלע .env טעקע. מען אימפארטירט אויך warnings, און מען שרייבט)"ignore"warnings.filterwarnings) כדי די ווארענונגען זאלן נישט ווערן געוויזן. אזוי גרייט מען צו די סביבה פונעם פראיעקט.

דערנאך קומט א קאוד וואס טוישט אויטאמאטיש איבער צו א נייערער ווערסיע פונעם LLM-מאדעל נאך א געוויסן דאטום. ער אימפארטירט דעם datetime מאדול כדי צו ארבעטן מיט דאטומס, און ער נעמט דעם היינטיגן דאטום. צום סוף אינסטאלירט מען דאס פעקל wikipedia, וואס מען וועט ניצן ווייטער.

אגענטן מיט LangChain און א הקדמה צו LLM-טשעטבאטן (11:55:20)

א Python-אגענט און געצייג פאר חשבון און Wikipedia (11:55:20)

אין דעם טייל פון קורס גייט מען ווייטער מיט דער LangChain -ארבעט. צום ערשט ווערט אימפארטירט create_python_agent, און דערנאך AgentType, initialize_agent, load_tools , דאס געצייג PythonREPLTool, PythonREPL, און ChatOpenAI.

מען לויפט אדורך דעם קאוד וואס שטעלט אויף דעם קאנפיגוראציע-וועג, און מען רופט ChatOpenAI . די temperature ווערט געשטעלט אויף נול, און אלס מאדעל ווערט אריינגעגעבן דער llm_model וואס איז פריער באשטימט געווארן.

דערנאך נוצט מען load_tools צו לאדן א סכום געצייג: llm-math און wikipedia. עס ווערט ערקלערט אז די שורה לאדנט די צוויי געצייג און פארבינדט זיי מיט דעם שפראך-מאדעל וואס איז פריער דעפינירט געווארן.

די קומענדיגע שורה שאפט א טשעט-אגענט מיט דער פונקציע initialize_agent. דער אגענט באקומט די געצייג llm-math און wikipedia, און ער ווערט קאנפיגורירט מיט א טשעט-טיפ פון) AgentTypeדער טיפ מיט "chat zero-shot").

צוויי פראבעס: א חשבון-פראגע און א פראגע וועגן א בוך (11:56:24)

די ערשטע פראבע איז די פראגע: וויפל איז 25% פון 300? וועז מען רופט דעם אגענט מיט דעם אינפוט, נוצט ער די געצייג און דעם שפראך-מאדעל. דער חשבון ווערט געטון דורך דעם llm-math געצייג, און דער אגענט גיט צוריק דעם ריכטיגן ענטפער.

די צווייטע פראבע: א וואריאבל מיטן נאמען question האלט א פראגע וועגן א בוך וואס Tom Mitchell האט געשריבן. ער איז אן אמעריקאנער קאמפיוטער-וויסנשאפטלער און א פראפעסאר אין Carnegie Mellon University. מען דערווארט אז דער אגענט וועט צוריקגעבן דעם נאמען פונעם בוך "Machine Learning", א באקאנט לערנבוך אין פעלד פון מאשין-לערנען. דער רעזולטאט ווערט געהאלטן אין א וואריאבל result.

וועז מען לויפט עס, זעט מען דעם גאנג פון דעם אגענט: א "thought", א "page" פון Wikipedia, און א "observation". צום סוף שרייבט דער אגענט: "I have found that Tom Mitchell wrote the book Machine Learning. I now know the final answer". דאס הייסט, ער האט געפונען דעם ענטפער און ווייסט אז דאס איז דער ענדגילטיגער ענטפער.

א Python-אגענט וואס סארטירט א קונדן-ליסטע (11:57:25)

דערנאך שאפט מען א Python-אגענט מיט create_python_agent. דער אגענט באקומט דעם באשטימטן שפראך-מאדעל, וואס טוט די ארבעט און גיט די ענטפערס. ער נוצט אויך דאס געצייג PythonREPLTool. דאס געצייג לאזט דעם אגענט לויפן Python-קאוד אין א REPL, א "read–eval–print loop": מען לייענט א שורה, מען רעכנט זי אויס און מען דרוקט דעם רעזולטאט, אין א ראד.

מען דעפינירט א ליסטע פון קונדן. יעדע אונטער-ליסטע שטעלט פאר איין קונה מיט צוויי עלעמענטן: דער ערשטער נאמען און דער פאמיליע-נאמען. אן ערשט ביישפיל איז Harrison און Chase, און אזוי ווייטער מיט די אנדערע קונדן.

די שורה דערנאך הייסט דעם אגענט סארטירן די קונדן-ליסטע: צום ערשט לויטן פאמיליע-נאמען, און דערנאך לויטן ערשטן נאמען. מען רופט די מעטאד run מיט א פארמאטירטן סטרינג, און די קונדן-ליסטע ווערט אריינגעשטעלט אינעם פראמפט.

דערנאך ווייזט מען ווי אזוי צו עפענען דעבאגינג אין LangChain. מען שטעלט:

True = langchain.debug

דאס גיט אויספירלעכע לאגס און אינפארמאציע וועגן יעדן שריט וואס דער LangChain-אגענט טוט. מיט דעם לויפט מען נאכאמאל די זעלבע סארטירונג, פאמיליע-נאמען צום ערשט און דערנאך ערשטער נאמען. אזוי זעט מען דעם גאנצן אינעווייניגן גאנג פון דעם אגענט.

אן אייגן געצייג: א פונקציע וואס גיט דעם היינטיגן דאטום (11:58:26)

דער קומענדיגער שריט איז צו בויען אן אייגן געצייג. מען אינסטאלירט datetime, און מען אימפארטירט date פון datetime און tool פון langchain.agents.

דער קאוד דעפינירט א געצייג-פונקציע מיטן נאמען time. זי ווערט באצייכנט מיט דעם @tool דעקאראטאר פון LangChain. די פונקציע נעמט איין פאראמעטער text, וואס מען דערווארט אז ער זאל זיין א ליידיגער סטרינג. זי גיט צוריק דעם היינטיגן דאטום אלס סטרינג. ביי דער פונקציע איז דא א docstring, א באשרייבונג אין קאוד. די באשרייבונג מאכט קלאר אז די פונקציע איז געמאכט נאר כדי צו געבן דעם היינטיגן דאטום, און אז מען דארף זי נוצן נאר פאר דעם צוועק.

דערנאך שאפט מען נאכאמאל אן אגענט מיט initialize_agent. דאס מאל באקומט ער סיי די פריערדיגע פארטיגע געצייג און סיי דאס נייע time געצייג, כדי ער זאל קענען געבן דעם היינטיגן דאטום. דער אגענט נוצט דעם באשטימטן שפראך-מאדעל צו טון די ארבעט און ענטפערן.

צום סוף פרובירט דער קאוד פרעגן דעם אגענט: "What's the date today?" מען דערווארט אז דער אגענט זאל ענטפערן מיטן היינטיגן דאטום דורכן נוצן דאס time געצייג. די רוף איז איינגעוויקלט אין א try-בלאק. אויב עס פאסירט א טעות, למשל א פראבלעם מיט דרויסנדיגן צוטריט אדער א געצייג וואס פעלט, גייט די פראגראם אריבער צום except-בלאק. דארט דרוקט זי די מעלדונג "exception on external access", און די מעלדונג ווייזט אז עפעס איז נישט געגאנגען גלאט ביים זוכן די אינפארמאציע. אזוי ווערט עס פארזיכערט אז דער פראצעס זאל נישט צוזאמענפאלן, און דער באניצער באקומט א קלארע מעלדונג אויב עס איז דא א פראבלעם.

מיט דעם ענדיגט זיך דער טייל "LangChain for LLM App Development". דער קורס מוטיגט אריינצוטון די קענטעניש אין פראקטיק: עקספערימענטירן מיט LangChain, פארבעסערן דעם שכל-גאנג פון די LLM-מאדעלן, און זוכן נייע וועגן צו פארשטארקן אויטאמאציע.

דער פלאן פונעם נייעם טייל: ווי אזוי צו בויען אן LLM-טשעטבאט (12:00:27)

דערנאך הייבט זיך אן א נייער טייל: ווי אזוי מען שאפט אן LLM-טשעטבאט. זיין סדר איז:

- א קורצע הקדמה צו LLM-מאדעלן און טשעטבאטן.
- אויפשטעלן די ארבעט-סביבה פאר אנטוויקלונג.
- זאמלען און צוגרייטן דאטא, אזוי אז LLM-מאדעלן זאלן זי קענען פארשטיין.
- אריינשטעלן LLM-מאדעלן אין דער ארבעט.
- צעשניידן די דאטא אין שטיקלעך ("chunking") און זיי פארוואנדלען אין embeddings, דאס הייסט וועקטארן, פאר וועקטאר-זוכונג אדער RAG (Retrieval Augmented Generation).
- באהאנדלען די פראגעס פונעם באניצער (query processing) און בויען רעקאמענדאציע-סיסטעמען.
- Zero-shot text classification, א סארט prompt engineering.

- צום סוף, sentiment analysis, דאס הייסט דערקענען די שטימונג פון א טעקסט, מיט LLM-מאדעלן. מען קען שאפן פארשידענע סארטן LLM-מאדעלן, אריינגערעכנט א מאדעל פאר sentiment analysis.

עס ווערט אויך צוגעזאגט אז צום סוף פון דער לעקציע וועט מען דורכגיין קאוד מיט LangChain. מיט דעם קאוד שאפט מען א טשעט-מאדעל וואס נוצט די OpenAI API, און מען שטעלט אים ארויס אין א פראדוקציע-סביבה מיט Streamlit.

וואס איז אן LLM? (12:02:43)

LLM הייסט "Large Language Model", א גרויסער שפראך-מאדעל. אזא מאדעל ווערט טריינירט אויף א ריזיגע מאסע טעקסט, און ער קען שאפן טעקסט וואס זעט אויס ווי א מענטש האט עס געשריבן. דער קורס ערקלערט אז דאס איז אן ענטוויקלטערע ווערסיע פון די פריערדיגע געצייג פון natural language processing, וואס האבן געארבעט אויף דיפ-לערנען נויראל-נעצן. יענע מאדעלן האבן גענוצט ארכיטעקטורן ווי RNN, LSTM, GRU און ענליכע. זיי האבן געקענט פארשטיין א טעקסט און ארויסברענגען זיין אינעווייניגן באדייט. די LLM-מאדעלן זענען אבער א שריט ווייטער: זיי פארשטייען נישט נאר דעם טעקסט, נאר זיי קענען אויך שאפן נייעם טעקסט לויט דער אינפארמאציע וואס מען גיט זיי.

זיי ווערן טריינירט אויף א ריזיגע מאסע דאטא, און עס ווערט באטאנט אז זיי האבן נישט קיין פריער-באשטימטע מקורות; זייער קענטעניש קומט פון דעם טריינירן.

ווי אזוי ארבעט אן LLM: טאקענס און סטעמינג (12:04:16)

אן LLM נעמט א גרויסע מאסע טעקסט און צעטיילט אים אין שטיקלעך, וואס מען רופט "טאקענס". אויף דער פשוטסטער מדרגה קען מען זיך פארשטעלן א טאקען ווי א ווארט: אין א פאראגראף פון 10, 20 אדער 1,000 ווערטער קען יעדעס ווארט זיין איין טאקען. אמאל רעכנט מען א גאנצן זאץ אלס איין טאקען. דאס הענגט אפ אין דער מעטאד פון מאדעלירן: צי מען נוצט זאץ-טאקענס צי ווארט-טאקענס.

אויף דער גרונט-מדרגה קען מען זיך פארשטעלן טאקענס ווי ווערטער וואס זענען דורכגעגאנגען "stemming" אדער "lemmatization". דאס זענען יסודות פון natural language processing, וואו מען שניידט אפ די ווערטער ביז זייער גרונט-שורש. אזוי באקומט מען די זעלבע וואורט-פארעם פאר ענליכע ווערטער.

ביישפילן: "breaking" ווערט "break"; "breaks" ווערט אויך "break"; "into" בלייבט "into". "transformer" קען ווערן "transform", און "architectures" קען ווערן "architect". דער געדאנק איז צו צווזאמענברענגען אלע ענליכע ווערטער אונטער איין שורש-וואורט. דאס רופט מען tokenization און stemming איז natural language processing.

פון ווערטער צו נומערן (12:06:51)

נאך דעם צעטיילן אין טאקענס, פארוואנדלט מען די ווערטער אין א וועקטאר-פארעם, דאס הייסט א נומערישע פארשטעלונג. דער קורס גיט א געמיינט ביישפיל: יעדעס וואורט אינעם גאנצן ווערטער-בוך באקומט א נומער. "a" איז נומער 1, "b" איז 2, "lamb" איז 3, און אזוי ווייטער. די נומערן קען מען שרייבן אין פארשידענע פארמען, למשל hex-קאוד, בינארישע פארעם אדער אן אנדער פארעם. ווייל די נומערישע פארשטעלונג פון דער דאטא מאכט די זאכן קלארער, קען דער מאדעל די ווערטער פארשטיין אין א נומערישער פארעם.

דער attention-מעכאניזם און טראנספארמער (12:07:54)

וועו מען האט די טאקענס, נוצט אן LLM דעם attention-מעכאניזם. עס ווערט דערמאנט דער באקאנטער פארשונג-ארטיקל "Attention Is All You Need", וואס איז ארויס אין 2017 פון Google Brain. דער ארטיקל באשרייבט דעם self-attention מעכאניזם, וואס איז דער יסוד פון אלע טראנספארמער-מאדעלן. LLM-מאדעלן נוצן אויך דעם טראנספארמער-צוגאנג, מיט א self-attention שיכט און masked multi-head attention שיכטן; דאס איז דער גרונט-פרינציפ פון אלע LLM-מאדעלן.

דער בוי האט אן encoder-בלאק און א decoder-בלאק. זיי ארבעטן צוזאמען כדי צו פארשטיין וואס איז דער אינפוט, און וועלכער טעקסט וואלט געקענט ארויסקומען פאר דעם אינפוט. דאס מאכט מעגלעך דער attention-מעכאניזם. דער קורס זאגט אז דעם self-attention מעכאניזם קען מען באהאנדלען אין פרטים אין א באזונדערן וויידעא.

עס ווערט ערקלערט אז דאס איז אן אנדערע דיפ-לערנען ארכיטעקטור, וואס נוצט נישט קיין נוירענען אויף דעם געווייגטלעכן אופן. ביי געווייגטלעכע דיפ-לערנען ארכיטעקטורן זענען פאראן נוירענען אין די באהאלטענע שיכטן (hidden layers); זיי נעמען דעם אינפוט, פראצעסירן אים און געבן ארויס א רעזולטאט. ביי LLM-מאדעלן ווערן די נוירענען פארביטן מיט self-attention בלאקן, וואס האבן צוויי וויכטיגע טיילן: אן encoder און א decoder. געווייסע LLM-מאדעלן האבן נאר אן encoder, אנדערע נאר א decoder, און אנדערע א צוזאמשטעל פון ביידע. לויט דעם קורס זענען די גענעראטיוע מאדעלן א צוזאמשטעל פון ביידע טיילן, encoding און decoding.

טשעטבאטן: א הקדמה (12:09:29)

צום סוף פון דעם אפשניט הייבט מען אן רעדן וועגן די סארטן טשעטבאטן. א טשעטבאט איז נאך א ביישפיל פון נוצן א גרויסן שפראך-מאדעל. מען שטעלט א פראגע, א פראמפט אדער א query פאר א מאשין, און די מאשין ענטפערט לויט דער פראגע.

צוויי סארטן טשעטבאטס און ווי מען בויט זיי (12:09:59)

רעגל-באזירטע און AI-באזירטע טשעטבאטס (12:09:59)

דער קורס ערקלערט אז טשעטבאטס ווערן איינגעטיילט אין צוויי סארטן: דער ערשטער איז א "רעגל-באזירטער" טשעטבאט (rule-based chatbot), און דער צווייטער איז אן AI-באזירטער טשעטבאט.

וואס איז א רעגל-באזירטער טשעטבאט? דאס איז א טשעטבאט וואס האט א פעסטגעשטעלטן סעט רעגלען און פריער-אנגעשריבענע סקריפטן פאר יעדע געווייסע פראגע. למשל, אויב מען האט שוין פריער באשטימט אז וועו איינער פרעגט "How are you?" זאל ער ענטפערן א געווייסן ענטפער, וועט ער פונקט דעם ענטפער געבן. אבער אויב א פראגע געפינט זיך נישט אין זיין "ווערטערבוך", וועט ער זאגן: "אנטשולדיגט, איך קען נישט ענטפערן אויף די פראגע, איך בין נישט טרייניירט אויף דעם טייל". אזעלכע ווערן גערופן רעגל-באזירטע טשעטבאטס אדער רעגל-באזירטע LLMs.

עס ווערט ערקלערט אז אזעלכע רעגל-באזירטע סיסטעמען ניצן געווענליך decision -אלגאריטמען אדער random forest-ארטיגע אלגאריטמען כדי צו פארשטיין און שאפן רעגלען און דערנאך געבן ענטפערס. דאס איז די לאגיק וואס שטייט הינטער זיי. היינט ניצט מען זיי שוין נישט אזוי פיל; היינט האט מען AI-באזירטע

טשעטבאטס, וועלכע זענען אסאך מער רעספאנסיוו און בעסער. אין דעם קלאס וועט מען ארבעטן מיט אן AI-באזירטן טשעטבאט.

די רעגל-באזירטע טשעטבאטס זענען זייער פשוט אויפצושטעלן און איינצופירן, אבער זיי קענען נישט באהאנדלען קאמפליצירטע פארמולירונגען, ווייל מען קען זיי נישט געבן גענוג ביישפילן. מיט אן AI-באזירטן טשעטבאט, אנדערש, קען מען אריינלייגן סיי וועלכע פראגע. מסתמא האבן רוב מענטשן שוין געגוצט AI-באזירטע טשעטבאטס ווי ChatGPT, Anthropic אדער Gemini – אלע זענען AI-באזירטע טשעטבאטס.

וויאזוי אויסקלייבן דעם ריכטיגן טשעטבאט (12:11:30)

דער קורס באטאנט אז וועלכער טשעטבאט מען דארף ניצן הענגט אפ אין די באדערפענישן, און מען דארף קוקן אויף א ריי פאקטארן:

- וויפיל קאסטאמיזאציע (צופאסונג) מען דארף.
- וואס איז די קאסט.
- וואס זענען די באדערפענישן פון) deploymentווי און וויאזוי מען שטעלט עס אויף).
- וויפיל מען וויל אז עס זאל זיין open source, און צי עס איז פארהאן אזא וועריע.
- דער לעצטער און וויכטיגסטער פאקטאר: וויפיל פריוואטע דאטא איז פארהאן – דאס הייסט דאטא וואס איז נישט עפנטלעך צוגענגלעך, אזוי אז OpenAI-מאדעלן אדער open source מאדעלן האבן זיך נישט געקענט דערפון לערנען. צי מען וויל אריינרעכענען אזא דאטא אדער נישט.

אויב מען וויל אריינרעכענען אזא דאטא, ענדערט זיך דער סארט AI-טשעטבאט. עס זענען פארהאן צוויי סארטן AI-באזירטע טשעטבאטס: איינער וואס ניצט פשוט פריער-טריינירטע (pre-trained) מאדעלן – מיט דעם וועט מען ארבעטן אין דעם קלאס; און דער צווייטער וואס ניצט פריער-טריינירטע מאדעלן און נעמט דערצו אויך א שטיק דאטא, אדער וועט טריינירט פון אנדערע דאטא. דאס רופט מען דער RAG -צוגאנג – retrieval augmented generation.

אויפשטעלן די ארבעטס-סביבה (12:13:01)

כדי צו שאפן א development environment קען מען אויסקלייבן פון אסאך זאכן; עס זענען פארהאן אסאך APIs. אין דעם אפטייל וועט מען ניצן דעם OpenAI API, וואס איז לייכט און פריי צוגענגלעך.

מען דארף א קאודירונג-סביבה – אין דער לעקציע ניצט מען Jupyter Lab. די ביבליאטעקן וואס מען וועט ניצן:

- Streamlit
- LangChain
- NumPy און Matplotlib

דאס וועט זיין גענוג. צי מען וויל א cloud-באזירטע deployment אדער) on-premisesאויף די אייגענע מאשינען) – דאס דארף מען פארשטיין, ווייל cloud-באזירטע deployments פאדערן מער רעסורסן. מען דארף זיך פארזיכערן אז מען האט דעם ריכטיגן API -צוטריט. אין דער לעקציע ניצט מען א פערזענלעכן OpenAI-צוטריט, און אין קאוד, אויף דעם ארט וואו מען דארף אריינלייגן דעם OpenAI key, וועט עס אנגעצייכנט ווערן.

מען דארף אויך א גרונט-פארשטאנד אין Python: וויאזוי מען שרייבט Python-סקריפטן, און וויאזוי מען פארבינדט APIs מיט די סקריפטן – צוויי וויכטיגע זאכן. מען וועט ניצן Jupyter Notebook און VS Code, אבער מען קען ניצן סיי וועלכע טעכנאלאגיע, ווייל מען וועט שרייבן א py-טעקע; סיי Jupyter Notebook , סיי VS Code, סיי א פשוטע Python-עדיטאר – וואס עס איז באקוועם.

API-צוטריט, געצייג און דאטא (12:14:34)

API-צוטריט קען מען בעטן פון OpenAI, Cohere אדער Hugging Face. אין דעם קלאס ניצט מען OpenAI. מען גייט צו platform.openai.com, מען רעגיסטרירט זיך, און מען קען גענערירן דעם אייגענעם OpenAI API key.

פאר די טשעטבאט-לאגיק קען מען ניצן transformers אדער LangChain – און דאס איז וואס מען וועט טון. LlamaIndex קען מען ניצן פאר דאטא-retrievalצוריקברענגען דאטא). FastAPI און Gradio קענען העלפן מיט דעם deployment-טייל, אבער דאס וועט מען נישט ניצן. מען קען ניצן cloud-סערוויסעס ווי AWS, Azure אא"וו – אבער דאס וועט מען אויך נישט באהאנדלען, ווייל עס איז נישט אינעם ראם.

וועגן זאמלען און צוגרייטן דאטא: מען דארף זאמלען די דאטא אין פארעם פון CSV, JSON אדער דאטא פון web scraping, און מיט NumPy און pandas שאפן און אויסרייניקן די דאטא. אבער אין דעם אפטייל וועט מען נישט ארבעטן מיט NLTK אדער spaCy, ווייל מען וועט דערווייל נישט טריינירן דעם מאדעל – מען ניצט פריער-טריינירטע מאדעלן פונעם OpenAI API.

אויסקלייבן דעם ריכטיגן מאדעל (12:15:36)

דער קורס גייט איבער צום פראקטישן טייל: וויאזוי שאפט מען LLM-טשעטבאטס. אויב מען וויל אליין טריינירן אן LLM-מאדעל, דארף מען א מאדעל-ארכיטעקטור, וואס מען קען אראפלאדן דורך transfer learning-מעטאדן. אלע אזעלכע מאדעלן קען מען באקומען פון Hugging Face , אדער גלייך פון דער transformers-ביבליאטעק, אדער פון TensorFlow אדער PyTorch – לויט וויאזוי מען איינפירט עס.

אויסער דעם איז וויכטיג וועלכער סארט מאדעל מען דארף: אויב מען טוט) sentiment analysisאנאליז פון געפילן אין טעקסט), דארף מען א sentiment analysis-מאדעל; אויב מען זוכט א פראגע-און-ענטפער צוגאנג, דארף מען אוא מאדעל; אויב מען דארף אלגעמיינע פראגעס און ענטפערס, קען מען ניצן אן אלגעמיינעם מאדעל ווי ChatGPT. דאס איז די ערשטע וויכטיגסטע אויפגאבע – אויסקלייבן דעם ריכטיגן מאדעל און די ריכטיגע APIs.

עס וועלט איבערגעחזרט א פארגלייך:

- פאר אלגעמיינע וועלסאטיליטעט – GPT איז גוט.
- פאר open source קאסטאמיזאציע – LLaMA.
- פאר עפעקטיווקייט – Mistral.
- פאר סקאלירבארקייט – Falcon.

דער ריכטיגער מאדעל הענגט אפ אין פערפארמענס, קאסט און deployment-באדערפענישן. מאדעלן קען מען באקומען פון Hugging Face, אדער ניצן API-באזירטע מאדעלן; אין דער לעקציע האט מען OpenAI API-צוטריט, און דארט איז פארהאן GPT-4o mini, וואס מען וועט ניצן.

וואס איז RAG – retrieval augmented generation (12:17:08)

כדי צו שאפן אן LLM-טשעטבאט דארף מען פארשטיין retrieval augmented generation. דער ביישפיל אין דער לעקציע: מען שרייבט אריין א פראגע, "What is the role of Python?". דער סיסטעם זוכט אינעם גאנצן קאנטעקסט וואס מען האט אים געגעבן, און געפינט וועלכע ווערטער זענען די מערסט רעלעוואנטע צו דער פראגע און איר באדייט. דערפון בויט ער א סטרוקטור פון די וויכטיגסטע שליסל-ווערטער. דערנאך פירט ער דורך א ברייטע זוכעניש אין דער דאטא וואס ער פארמאגט, געפינט פאסנדע שליסל-ווערטער און ענטפערס פון פארשידענע מקורות, לערנט זיך פון אלע מקורות, און קומט צוריק מיט א לייזונג.

דער נוצן פונעם RAG-צוגאנג: דער מאדעל וועט נישט געבן צופעליגע, אומקלארע ענטפערס, און וועט נישט געהערירן ענטפערס אן קיין קאנטעקסט; ער וועט שטענדיג זוכן סעמאנטישע אינפארמאציע און געבן גוטע, קאנטעקסטועלע ענטפערס.

chunking, embedding און vector search (12:18:42)

דאס זעלבע געשעט ביי chunking און embedding אין vector search: מען צעברעכט דעם גאנצן טעקסט אויף קליינע, באהערשבארע שטיקלעך, וואס ווערן געווען tokens ; די ווערן איבערגעביטן אויף נומערן, געווען vectors. דאס קען מען טון מיט אסאך ביבליאטעקן:

- דער character text splitter פון LangChain פאר chunking.
- NLTK.
- tiktoken – עס ווערט אונטערגעשטראכן אז מען זאל דאס נישט פארמישן מיט TikTok; עס איז א גאנץ אנדער זאך, א ביבליאטעק וואס צעטיילט טעקסט אויף tokens.
- sentence transformers, וועלכע שאפן די vectors – א נומערישע פארשטעלונג פונעם טעקסט.

דערנאך היט מען דאס אפ אין א דאטאבאזע – א פשוטע דאטאבאזע איז געווג. נאכדעם זוכט מען similarity-חשבונות, ווי cosine similarity, Euclidean distance אדער dot product , כדי צו געפינען וואס פאסט. RAG ניצט פונקט דעם צוגאנג: ער געפינט ענליכע קאנטעקסט-ווערטער און ברענגט ענליכע קאנטעקסטועלע אינפארמאציע.

פראצעסירן פראגעס (12:19:42)

ביי query processing: מען פרעגט א פראגע, דער LLM צעברעכט זי אין טעקסט, פון טעקסט מאכט ער vectors, און ווייל עס איז א פראגע ווערט עס א פראגע-און-ענטפער צוגאנג. ער געפינט, למשל, 10 ענטפערס און פילטערט ארויס דעם בעסט-פאסנדן ענטפער פאר דער פראגע.

zero-shot טעקסט-קלאסיפיקאציע (12:20:14)

אין zero-shot text classification טריינירט מען נישט דעם מאדעל ספעציעל, און מען לייגט נישט צו קיין נייע רעסורסן. מען האט א פריער-טריינירטן מאדעל ווי GPT, וואס איז טריינירט אויף זייער אלגעמיינע ידיעות, וואס זענען אפשר נישט ספעציפיש צום קאנטעקסט פון דער פראגע. דער ביישפיל: ChatGPT איז טריינירט כמעט אויף אלעס אויף דער וועלט, אבער נישט אויף די ספעציפישע ענינים פון א געוויסער פירמע. מען קען אים פונדעסטוועגן פרעגן וועגן דער פירמע – ער וועט נישט געבן קיין ספעציפישן ענטפער, אבער ער וועט געבן אן אלגעמיינעם ענטפער. דאס איז zero-shot: מען טריינירט נישט דעם LLM אויף ספעציפישער דאטא, נאר מען ניצט א פריער-טריינירטן מאדעל און פרעגט אים וועגן נייער דאטא.

געפינען קלאסיפיקאציע-מאדעלן אויף Hugging Face (12:21:16)

אין דער לעקציע ווערט שנעל געוויזן וואו מען געפינט פריער-טריינירטע קלאסיפיקאציע-מאדעלן: מען גייט צו huggingface.co, דארט זענען פארהאן אסאך אזעלכע מאדעלן. מען גייט צו Models, און מען קליקט אויף דער אויפגאבע (task) וואס מען זוכט – למשל אונטער natural language processing, Zero-Shot Classification. דארט זעט מען א רייַ מאדעלן ספעציעל פאר zero-shot. איינער פון זיי איז bart-large-mnli פון Facebook. מען קען גיין צו "Files and versions" און זען דעם גאנצן מאדעל, און ליינען די README.md-טעקע כדי צו געפינען אינפארמאציע וועגן אים. ווייל דאס איז ווי א GitHub, קען מען clone-ן דעם גאנצן repository גלייך פון Hugging Face און עס ניצן לאקאל אויפן אייגענעם קאמפיוטער. סיי וועלכער מאדעל קען געוצט ווערן; bart-large-mnli איז איין ביישפיל.

pipelines, מעסטונגען און sentiment analysis (12:22:48)

אין zero-shot קלאסיפיקאציע קען מען טון נאך אסאך זאכן. מען קען שאפן transformer pipelines, וואו מען פירט דורך איין טראנספארמאציע, נאכדעם א צווייטע, נאכדעם א דריטע, אזוי ווייטער. מען קען אויך ניצן מעסטונגען פון scikit-learn, ווי precision, recall, F1 scores אדער כדי צו קאנטראלירן צי די קלאסיפיקאציע איז ריכטיג. צום סוף גייט דער קורס איבער צו sentiment analysis מיט LLMs, און ערקלערט אז LLMs קענען זיין זייער נוצלעך אויך פאר sentiment analysis.

סענטימענט-אנאליז און א LangChain טשעטבאט מיט Streamlit (12:23:18)

סענטימענט-אנאליז: פארשטיין דעם געפיל אין א טעקסט (12:23:18)

דער קורס ענדיגט דעם טעארעטישן טייל מיט סענטימענט-אנאליז. עס ווערט ערקלערט אז ביי סענטימענט-אנאליז פרובירט מען אויסצוגעפינען דעם געפילישן טאן פון א טעקסט, כדי טשעטבאטס זאלן קענען פארשטיין וואס דער באניצער פילט. מען טריינירט די טשעטבאטס אויף דאטא וואס אנטהאלט געפילישע זייטן, און מען צייכנט אן צי עס איז א פאזיטיווער געפיל, א נעגאטיווער, א טרויעריגער אדער א פרייליכער.

לויט דעם טריינירן וועט דער LLM קענען זאגן וועלכער געפיל שפיגלט זיך אפ אין א געוויסן טעקסט. עס ווערט אונטערגעשטראכן: דער מאדעל וועט נישט קענען נאכמאכן דעם געפיל, אבער ער וועט קענען זאגן וועלכער געפיל איז אויסגעדריקט אינעם קאנטעקסט. דאס קען זיין מיט אודיא, מיט ווידעא, אדער מיט טעקסט.

מען דערמאנט אז DistilBERT איז א זייער גוטער קלאסיפיקאציע-מאדעל פאר סענטימענט-אנאליז. א סענטימענט-אנאלייזער געפינט מען אויך ביי אנדערע מאדעלן, ווי DistilBERT, RoBERTa און Llama. עס איז איינע פון די קלאסן אין א פריי-טריינירטן מאדעל, וואס מען קען גלייך ניצן צו אנאליזירן געפילן. דערמיט ענדיגט זיך דער טעארעטישער אריבערבליק איבער LLMs, און מען גייט איבער צו דער פראקטישער אויספירונג.

אנהייבן די דעמא: Anaconda, JupyterLab און א Python-פייל (12:24:52)

אין דער לעקציע עפנט מען Anaconda Navigator, און פון דארט עפנט מען די JupyterLab-סביבה. מען האט שוין א צוגעגרייטן סקריפט, און מען גייט אים דורך שורה נאך שורה און ערקלערט וואס יעדע שורה טוט.

די ציל פון דער דעמא איז צו בויען א טשעטבאט אויף דעם שטייגער פון ChatGPT, דורכן OpenAI API. אזא טשעטבאט קען מען לויפן אויף די אייגענע לאקאלע סערווערס אדער ביי זיך. דאס איז אין עיקר א מין קאפיע פון ChatGPT וואס מען קען אליין אויפשטעלן. מען זאגט אז דער פייל וועט זיין צוגעשטעלט אין דער באשרייבונג פונעם ווידעא.

כדי צו שאפן דעם סקריפט איז מען געגאנגען צו File, דערנאך New, און האט אויסגעקליבן א Python-פייל.

די נויטיגע ביבליאטעקן און וואס יעדע טוט (12:26:22)

פאר מען אימפארטירט די ביבליאטעקן, דארף מען זיי אינסטאלירן. אין קאוד שרייבט מען דאס ארויף ווי א באמערקונג: מען דארף לויפן) pip installאדער, אויב מען ניצט א באזונדערע סביבה, אינסטאלירן זיי אין יענער סביבה). די ביבליאטעקן זענען:

- langchain: צו בויען די לאגיק פונעם טשעטבאט.
- streamlit: צו מאכן דעם באניצער-אינטערפייס, א וועב-בלאט.
- openai: פאר דער פארבינדונג מיט OpenAI.

עס ווערט ערקלערט אז מען וועט זיך נישט אריינלאזן צו טיף אינעם דיזיין. מען נעמט א פשוטן אינטערפייס, ווייל דער ציל איז נישט צו לערנען ווי אזוי צו פלאנירן א וועב-בלאט. Streamlit איז אינגאנצן א באזונדערע ביבליאטעק, און מען קען אויף איר אפהאלטן א באזונדערע לעקציע.

דערנאך גייען די אימפארטן איינס נאך דאס אנדערע:

- import streamlit as st: אימפארטירט Streamlit אונטערן קורצן נאמען st.
- פון langchain.chat_models אימפארטירט מען דעם קלאס ChatOpenAI. דער סוב-מאדול לאזט זיך פארבינדן מיט OpenAI-סערווערס און ניצן דעם API -שליסל. דער קלאס דינט צו שאפן און האלטן די OpenAI-געצייג.
- פון langchain.chains אימפארטירט מען ConversationChain. דאס לאזט שאפן א קייט פון שמועסן. ווען מען פרעגט א שאלה און באקומט א תשובה, קען מען דערנאך פרעגן א ווייטערדיגע, טיפערע שאלה, און דער מאדעל געדענקט די פריערדיגע שאלות אינעם קאנטעקסט פונעם איצטיגן שמועס. אנשטאט "איין שאלה, איין תשובה, און פארטיג", קען מען ממשיך זיין דעם שמועס אן אויפהער.
- פון langchain.memory אימפארטירט מען ConversationBufferMemory. דאס לאזט געדענקען דעם פריערדיגן שמועס. אויב מען האט אנגעהויבן מיט איין שמועס און מען וויל ווייטער גיין מיט נאך איינעם, קען מען דעם פריערדיגן שמועס אפהיטן, ניצן און ווידער אויפלאדן.
- os: מען ניצט עס פאר אן אלגעמיינעם צווועק, צו שאפן environment variables . אויב מען ניצט נישט קיין environment variables, קען מען os אינגאנצן אויסלאזן.

אפהיטן דעם OpenAI API-שליסל זיכער (12:29:28)

דער ערשטער שריט איז צו שטעלן דעם OpenAI API -שליסל אויף א זיכערן אופן. מען שאפט א ווערייאבל openai_api_key און מען נעמט דעם ווערט פון Streamlit secrets.

עס ווערט ערקלערט וואו די Streamlit secrets געפינען זיך: געוועלך האלט מען זיי אין א TOML-פייל. אינעם דיפאלט-פאלדער פונעם באניצער-חשבון, וואו Anaconda איז אינסטאלירט, געפינט מען א פאלדער

מיטן נאמען .streamlit, און דארט אינעווייניג א פייל מיטן נאמען secrets.toml. אויב דער פייל איז שוין דארט, שרייבט מען דעם שליסל אריין. אויב נישט, דארף מען שאפן א נייעם טעקסט-דאקומענט:

- אינעווייניג שרייבט מען: OPENAI_API_KEY = "..." , און צווישן די געדזעפיסלעך לייגט מען אריין דעם אייגענעם API-שליסל.
- מען היט אפ מיט Save As, מען קלייבט אויס All Files, און מען גיט דעם פייל דעם נאמען secrets.toml.

לויטן קורס לאזט דאס אריבערפירן דעם שליסל צום מאדעל און איבערן נעץ מיט ענקריפציע. אין דער דעמא איז דער פייל secrets.toml שוין גרייט און אנטהאלט דעם API-שליסל.

אינעם קאוד הייסט מען Streamlit גיין צו די secrets און זוכן דעם OpenAI API-שליסל. אויב ער איז דארט, נעמט מען אים פון דארט. אויב נישט, נעמט מען אים פון דעם environment variable, דורך os.getenv("OPENAI_API_KEY"). וועט עס וויל נישט ניצן secrets, קען שטעלן דעם וועריאבל גלייך אינעם קאוד מיט os.environ["OPENAI_API_KEY"] = "..." און דארט אריינלייגן דעם שליסל. מען צייכנט אן אז דאס איז א ברירה, און מען לייגט עס אריין ווי א באמערקונג.

איניציאליזירן דעם LLM: טעמפעראטור און מאדעל-נאמען (12:32:05)

נאכדעם איניציאליזירט מען דעם LLM:

llm = ChatOpenAI(openai_api_key=openai_api_key, temperature=..., model_name="gpt-4")

ChatOpenAI איז די פונקציע וואס פארבינדט מיט די OpenAI-סערווערס. אלס openai_api_key גיט מען איבער דעם וועריאבל וואו מען האט אפגעהיט דעם שליסל. מען דערמאנט זיך אויב מען האט גארנישט אנדערש גרייט, קען מען דעם שליסל גלייך אריינקלעבן אינעם קאוד, אן צו שאפן עפעס. אבער עס וועלט געווארנט: אויף אזא אופן לייגט דער שליסל אפן, און מען וויל אים דאך מסתמא האלטן געהיים.

דער temperature באשטימט ווי פיל צופעליגקייט מען וויל אין די תשובות. וואס העכער די צאל, אלץ מער צופעליגקייט ברענגט מען אריין אין די תשובות. דער קורס זאגט אויך אז וואס גרעסער דער וועלט, אלץ בעסער וועלן זיין די תשובות.

מיט model_name קלייבט מען דעם מאדעל, לויט וועלכע מאדעלן מען האט צוטריט צו אין OpenAI. אין דער דעמא האט מען צוטריט צו GPT-4, און מען שרייבט דאס. אנדערש קען מען אויסקלייבן GPT-3.5 Turbo אדער וועלכן מאדעל מען האט צוטריט צו.

זכרון און די שמועס-קייט אין Streamlit (12:33:07)

דערנאך שטעלט מען אויף דעם זכרון אויף דער Streamlit-פלאטפארמע. די ציל איז: אויב עס איז פאראן א פריערדיגער זכרון, זאל מען אים צוריקברענגען. אויב נישט, הייבט מען אן דעם שמועס פון ניי.

מען ניצט st.session_state, א וועריאבל וואס האלט אפ וועלט. מען קוקט צי "memory " געפינט זיך אין session_state. אויב יא, איז דער סימן true, און אויב נישט, איז ער false. דער קאוד זאגט: אויב memory איז נישט אין session_state, שאפט מען st.session_state.memory און מען גיט אים א וועלט מיט ConversationBufferMemory(). דער זכרון איז דעמאלט נאך ליידיג.

נאכדעם שאפט מען דעם שמועס מיט ConversationChain:

- llm: דער LLM-מאדעל וואס מען האט פריער איניציאליזירט.

- memory: דער איצטיגער זכרון-מצב, וואס איז אין אנהייב ליידיג. ווי ווייטער דער שמועס גייט, ווערט ער דערהיינטיגט.
- verbose=False: דאס זארגט אז ווארענונגען אדער צווישן-מעלדונגען זאלן נישט ווערן געוויזן.

דער אינטערפייס און דער באניצער'ס אינפוט (12:35:12)

יעצט מאכט מען דעם אינטערפייס. מיט st.set_page_config באשטימט מען דעם וועב-בלאט אינעם בלעטערער: דער טיטל פונעם בלאט איז "LLM Chatbot", און מען קען אויך לייגן א page_icon , למשל א GIF-בילד אדער וואס מען וויל. מען קען דאס אויך אויסלאזן. אין דער דעמא ניצט מען א פשוטן סימבאל, ווייל מען האט נישט קיין קליינע בילדלעך בנמצא.

דערנאך שטעלט מען מיט st.title דעם טיטל וואס ערשיינט אינעם צענטער פונעם ערשטן בלאט: "LangChain Chatbot by Simplilearn". די רעזולטאט איז א וועב-בלאט מיטן טיטל LLM Chatbot אינעם בלעטערער, דעם אויסגעקליבענעם אייקאן, און דער גרויסער טיטל אינמיטן פונעם עקראן.

דערנאך לייגט מען אוועק א ליידיג קעסטל צו באקומען דעם באניצער'ס אינפוט:

user_input = st.text_input("You:", key="input")

st איז די Streamlit-ביבליאטעק. text_input ווייזט אן א קעסטל מיטן וואָרט "You:", און מיט key="input" ווייסט מען אז אט דא דארף מען אריינשרייבן דעם אינפוט.

אויב מען באקומט א אינפוט פונעם באניצער, איז די תשובה:

response = conversation.predict(input=user_input)

conversation איז די שמועס-קייט וואס מען האט פריער געשאפן. לויט דעם אינפוט שאפט זי א תשובה. נאכדעם וואס מען באקומט די תשובה, לייגט מען זי צו צום שמועס-זכרון מיט פאר-דעפינירטע פונקציעס: st.session_state.memory.chat_memory.add_user_message(user_input) לייגט צו דעם באניצער'ס מעסעדזש, און add_ai_message(response) לייגט צו די תשובה פונעם AI.

פארענדיקן דעם Streamlit טשעטבאט און אריינפירן LangChain (12:37:13)

ארויסווייזן דעם ענטפער און די שמועס-היסטאריע (12:37:13)

דער קורס האלט ביים סוף פון דעם קאוד פאר דעם טשעטבאט. נאך די צוויי פונקציעס, ווערט געשריבן דער ענטפער אויפן עקראן. דערפאר באניצט מען זיך מיט st.write. מען שרייבט א פארמאטירטע string , מיט markdown. דאס וואָרט "bot" ווערט ארויסגעווויזן אין דיקע אותיות, און דערנאך קומט דער ענטפער. וואס פאר א ענטפער עס זאל נאר קומען פונעם LLM מאדעל, וועט ער דארט ערשיינען.

צום סוף ווערט צוגעלייגט א טייל וואס איז אינגאנצן נישט נייטיג. עס איז אן אפציע פאר די וואס ווילן זען די פריערדיגע שמועסן. מען שאפט א checkbox מיט st.checkbox, מיט דעם טעקסט "show chat history". ווען מען צייכנט אן דעם קעסטל, ווערט דער ווערט true. ווען נישט, בלייבט ער false.

אויב עס איז true, גייט מען דורך יעדע message וואס ליגט אין st.session_state.memory.chat_memory.messages. אין session state ליגט שוין דער chat memory, און דארט זענען אריינגעלייגט געווארן סיי די באניצער-מעסעדזשעס און סיי די AI-מעסעדזשעס. מען נעמט

ארויס אלע מעסעדזשעס. ביי יעדן ווייזט מען די ראלע: אויב דער type איז human, שרייבט מען "You", אנדערש "bot". דערנאך ווייזט מען דעם גאנצן אינהאלט. דערמיט איז פארענדיקט דער גאנצער טייל פון שאפן דעם LLM טשעטבאט.

לויפן די אפליקאציע אין בראוזער (12:38:13)

מען דארף אפהיטן די טעקע אונטערן נאמען chatbot.py. דערנאך עפנט מען א terminal. אין דער לעקציע איז דאס דער Windows PowerShell terminal. מען לויפט דעם באפעל:

```
streamlit run chatbot.py
```

נאכן דריקן Enter זאגט דער terminal אז מען קען שוין זען די Streamlit אפליקאציע. די אפליקאציע הייבט זיך אן אין הינטערגרונט, און זי עפנט זיך אליין אין בראוזער. דער terminal ווייזט א local URL: http://localhost:8505. ער ווייזט אויך א network URL מיט אן IP-אדרעס. מען קען זיך באניצן מיט ביידע.

אין בראוזער זעט מען דעם טיטל פונעם פענצטער, "LLM chatbot". עס איז געווען געפלאנט אז אן emoji זאל זיך באווייזן, אבער ער איז נישט ארויסגעקומען. אויפן בלאט זעט מען דעם נאמען פונעם טשעטבאט, "LangChain chatbot". דארט איז דא א קעסטל פאר "You" אריינצושרייבן א פראגע, און די אפציע "show chat history".

אויספרובירן דעם טשעטבאט (12:40:16)

מען פרעגט ערשט א גאר אלגעמיינע פראגע: ווער וועט געווינען דעם Champions League 2025 . נאכן דריקן Enter קומט אן ענטפער, אויב אלעס ארבעט און דער API לויפט. דער מאדעל ענטפערט אז ער קען נישט פאראויסזאגן די צוקונפט. ער ווייסט נישט ווער וועט געווינען, ווייל דער טורניר איז פול מיט טאלאנטירטע קבוצות, און אין פוטבאל קען אלעס פאסירן. מען קען אים פרעגן אנדערע שאלות.

דערנאך פרעגט מען א נייע פראגע: ער זאל געבן די נעמען פון אלע קבוצות וואס נעמען אנטייל. דער קורס באמערקט אז דער ענטפער הענגט אפ ווי פיל אינפארמאציע איז פאראן, און ווי מען שרייבט דעם פראמפט. דער מאדעל זאגט אז ער ווייסט נישט גענוי. אבער די בעסטע קבוצות פון פארשידענע ליגעס קעמפן פאר א פלאץ: Premier League, La Liga, Serie A, Bundesliga און Ligue 1, צוזאמען מיט ווייניגער באקאנטע ליגעס. מען זעט אויך נעמען פון געוויסע קבוצות.

ווען מען צייכנט אן "show chat history", זעט מען אונטן די גאנצע היסטאריע פון אלעס וואס איז געטון געווארן אין דער שמועס. נאכן אפצייכענען פארשווינדט עס ווידער. דער קורס ווארנט אז די אפליקאציע לויפט אויפן לאקאלן סערווער. דעריבער הענגט די שנעלקייט אפ פון די hardware פון דער מאשין, און עס קען זיין שנעל אדער פאמעלעך.

Deploy פון דער אפליקאציע (12:41:49)

דער נעקסטער שריט וואס מען קען טון איז deploy, דאס הייסט ארויסשטעלן די אפליקאציע אויף דער וועלט. אויפן בלאט איז דא א Deploy קנעפל, מיט עטלעכע ברירות:

- Streamlit Community Cloud
- Snowflake
- אייגענע (custom) פלאטפארמעס

אויב מען קלייבט Community Cloud און מען דריקט Deploy, דארף מען צוטריט צו א GitHub חשבון. מען דארף ארויפלייגן דעם קאוד אויף GitHub, און ערשט דערנאך קען מען מאכן deploy. דער קורס זאגט אז דאס וועט ערקלערט ווערן אין א באזונדערער לעקציע. מען בעט אויך אז ווער עס האט שאלות, זאל זיי שרייבן אין די קאמענטארן.

וואס איז LangChain? (12:42:19)

דא הייבט זיך אן ערקלערונג פון וואס LangChain איז. LangChain איז א framework וואס העלפט פראגראמירער בויען אפליקאציעס וואס ארבעטן מיט גרויסע שפראך-מאדעלן (LLMs). פשוט געזאגט, איז עס ווי א בריק צווישן דעם LLM און דער פאקטישער ארבעט וואס מען ווייל אז די אפליקאציע זאל טון.

ווען מען באניצט זיך גלייך מיט א מאדעל, גיט מען אים א פראמפט און מען באקומט צוריק אן ענטפער. אבער ריכטיגע אפליקאציעס דארפן פיל מער: זכרון (memory), דאקומענטן, APIs, געצייג (tools), דאטא-באזעס, עטלעכע שריט נאכאנאנד, און באשלוס-מאכן. LangChain העלפט ארגאניזירן דאס אלעס.

דער קורס גיט א משל: דער LLM איז דער מוח, און LangChain איז די סיסטעם וואס זאגט דעם מוח וואס צו טון. זי זאגט אים וועלכע אינפארמאציע צו נוצן, וועלכן געצייג אויפצורופן, און וואס צו טון דערנאך.

א ביישפיל: מען וויל בויען א טשעטבאט וואס ענטפערט שאלות פון די PDF טעקעס פון א פירמע. דער LLM אליין קען דאס נישט טון אויף א פארלעסלעכן אופן, סיידן מען גיט אים דעם אינהאלט. LangChain העלפט פארבינדן דעם מאדעל מיט יענע דאקומענטן. זי געפינט די ריכטיגע אינפארמאציע און שאפט א געהעריגן ענטפער. דער קורס באטאנט אז LangChain איז נישט דער שפראך-מאדעל אליין. עס איז א framework וואס העלפט בויען נוצלעכע, סטרוקטורירטע אפליקאציעס פאר דער ריכטיגער וועלט.

פארוואס LangChain איז וויכטיג (12:43:53)

היינט נוצן אסאך מענטשן נישט בלויז LLMs פאר א פשוטן שמועס. זיי בויען ערנסטע פראדוקטן דערמיט. אזעלכע פראדוקטן דארפן זיין געזויער, נוצלעכער, און מער פארבונדן מיט דאטא פון דרויסן. דא ווערט LangChain גאר וויכטיג.

א פשוטער אויפרוף צו א LLM ארבעט גוט פאר פשוטע ארבעטן, ווי סיכומים, איבערשרייבן, אדער שאפן טעקסט. אבער ווען מען וויל אז די אפליקאציע זאל טון עפעס קליגערס, הייבן זיך אן פראבלעמען. דער קורס רעכנט אויס עטלעכע אזעלכע פעלער:

- מען וויל אז דער מאדעל זאל געדענקען דעם קאנטעקסט פון פריערדיגע שמועסן.
- מען וויל אז ער זאל זוכן אין א דאקומענט איידער ער ענטפערט.
- מען וויל אז ער זאל זיך באניצן מיט געצייג, ווי א קאלקולאטאר, זוכן אויפן וועב, אדער דאטא-באזעס.
- מען וויל א workflow וואו איין שריט שפייזט דעם נעקסטן.
- מען וויל אן אגענט וואס קען אליין באשליסן וועלכע אקציע צו נעמען.

דאס אלעס מיט די הענט טון וועלט גאר שנעל א בלאָנושענדיש. LangChain גיט א סטרוקטור צום גאנצן פראצעס. זי שטעלט צו פארטיגע קאמפאנענטן פאר פראמפטס, ארויפלאדן דאקומענטן, retrieval (ארויסנעמען אינפארמאציע), output parsing, memory, chains און אגענטן. אנשטאט בויען אלעס פון אנפאנג, קען מען צוזאמשטעלן די שטיקער און שאפן א קלוגן workflow אסאך שנעלער. דערפאר ווערט LangChain אזוי פאפולער: זי לאזט גיין ווייטער פון בלויז פראמפטן, צו בויען ריכטיגע אפליקאציעס.

ביישפילן פון באנוץ (12:45:24)

איינער פון די מערסט פארשפרייטע באנוצן איז א סיסטעם פאר ענטפערן שאלות פון דאקומענטן. וועו א פירמע האט הונדערטער PDFs, פאליסיס, באריכטן אדער מאנואלן, דארף מען נישט אלעס איבערליינען. מען קען בויען א סיסטעם וואו דער באניצער פרעגט א פראגע אין פשוטן ענגליש, און די אפליקאציע געפינט דעם פאסיגן אינהאלט און ענטפערט.

אנדערע ביישפילן וואס דער קורס ברענגט:

- א טשעטבאט פאר קונה-שטיצע. ער ענטפערט שאלות איבער פראדוקטן פון אינערליכע וויסן-באזעס, געדענקט דעם קאנטעקסט פונעם באניצער, און קען אפילו איבערגעבן העכער וועו עס איז נויטיג.
- א געהילף פאר דורכקוקן resumes. ער ליינט resumes, פארגלייכט זיי מיט דער ארבעט-באשרייבונג, מאכט א סיכום פון די קאנדידאטן, און ראנגירט זיי לויט ווי זיי פאסן.
- א געהילף פאר פארשונג. ער זאמלט אינפארמאציע, מאכט סיכומים פון לאנגע טעקסטן, פארגלייכט מקורות, און שאפט סטרוקטורירטע רעזולטאטן.
- אן AI אגענט workflow. למשל, אן אגענט פאר פלאנירן נסיעות. ער פארשטייט וואס דער באניצער וויל, זוכט ערטער, שלאגט פאר א מארשרוט, שאצט אפ דעם בודזשעט, און ארגאניזירט דעם רעזולטאט שריט ביי שריט.
- געהילפן פאר שרייבן e-mail-ס, סיכומים פון באגעגנישן, שאפן SQL queries, און קאוד-געהילפן.
- לויט דעם קורס קען מען עס אויך נוצן פאר עצות אין רעכטלעכע און פינאנציעלע ענינים, דורך אנאליזירן דאקומענטן.

אין קורצן: LangChain איז נוצלעך יעדעס מאל וואס מען וויל אז א LLM זאל טון מער ווי בלויז ענטפערן אויף איין פראמפט.

אינסטאלירן Python און א פראגראם צו שרייבן קאוד (12:47:13)

איצט הייבט מען אן צו שטעלן LangChain. פריער דארף מען זיין זיכער אז Python איז אינסטאלירט. אין דער לעקציע באניצט מען זיך מיט VS Code, אבער PyCharm איז אויך א געצייג (IDE) וואס דער קורס האלט פון.

צו אינסטאלירן Python, שרייבט מען אין Google "python download" און מען גייט צום אפיציעלן וועבזייטל, python.org. די לעצטע ווערסיע דארט איז 3.14.3. זי איז פאראן פאר Windows, Linux, Mac OS, Android און נאך. מען נעמט אראפ דעם Python install manager, מען עפנט אים, און מען פאלגט די שריט. דער פראצעס איז גאר פשוט: מען דארף בלויז אנצייכענען די נויטיגע קעסטלעך.

פאר שרייבן קאוד זענען דא צוויי ברירות. די ערשטע איז PyCharm, פון דער פירמע JetBrains. מען קען נעמען די community ווערסיע. דער וועבזייטל דערקענט אליין וועלכע operating system מען נוצט, Windows, Mac OS, Linux אדער Ubuntu, און ער וויזט דעם פאסיגן אראפנעם. די צווייטע ברירה, פאר די וואס נוצן נישט PyCharm, איז Visual Studio Code. מען קען אים אראפנעמען גלייך פון זיין וועבזייטל. אין דער לעקציע איז VS Code שוין אינסטאלירט.

קאנטראלירן די וועדסיע און שאפן א virtual environment (12:49:17)

נאכן אינסטאלירן Python איז גאר וויכטיג צו גיין צום command prompt און קאנטראלירן אז Python איז אינסטאלירט געווארן ווי עס דארף צו זיין. דאס טוט מען דורך קוקן די וועדסיע:

version-- python3

אין דער לעקציע ניצט מען python3 אלס דעם באפעל סיי פאר קאנטראלירן סיי פאר לויפן פראגראמען. דער רעזולטאט ווייזט 3.12.5. דאס איז אן עלטערע ווערסיע, ווייל זי איז אינסטאלירט געווארן שוין פריער, אבער זי ארבעט אויך גוט מיט אלעס.

דערנאך שאפט מען א פאלדער, למשל מיטן נאמען "langchain project", און מען עפנט אים אין VS Code. דער ערשטער שריט דארט איז צו שאפן א virtual environment, מיט דעם באפעל:

m venv- python3

אויב מען וויל זען די אפיציעלע דאקומענטאציע, זוכט מען "create virtual environment python doc". מען קומט אן צום אפיציעלן Python וועבזייטל וואס ערקלערט ווי צו שאפן א virtual environment. דער קורס רעקאמענדירט עס אלס א גוטן מקור, וואו מען ערקלערט אויך אנדערע פאראמעטערס וואס מען דארף באטראכטן. דער אלגעמיינער באפעל איז m venv- python, און דערנאך גיט מען דעם וועג (path) וואו דער environment זאל געשאפן ווערן.

LangChain: אינסטאלאציע, Chains, פראמפטס און RAG (12:51:20)

דער ווירטועלער סביבה און די ליבראריס וואס מען דארף (12:51:20)

דער קורס הייבט אן מיטן שאפן א ווירטועלע סביבה (virtual environment). וועז מען טיפט דעם באפעל, זעט מען אז עס ווערט געשאפן א פאלדער מיטן נאמען venv. עס זענען דא נאך אפציעס, אבער מען דארף זיי נישט.

דער נעקסטער שריט איז אקטיווירן די ווירטועלע סביבה מיטן באפעל:

activate\Scripts\venv

דערנאך דארף מען אינסטאלירן עטליכע ליבראריס. צו ערשט ווערט געגעבן א קורצער איבערבליק:

- OpenAI – צו פארבינדן זיך מיט OpenAI מאדעלן ווי GPT 5.4 אדער GPT 5.2.
- FAISS – פאר וועקטאר-זוכן (vector search).
- ChromaDB – צו אפהיטן און ארויסנעמען embeddings.
- tiktoken – פאר פראצעסירן לויט טאקענס.
- pypdf – צו לאדן PDF דאקומענטן.
- BeautifulSoup 4 – אויב מען וויל "סקרעיפן" אדער לאדן אינהאלט פון וועבזייטלעך.
- sentence-transformers – פאר אנדערע embedding ליבראריס, אויב מען דארף.

LangChain און langchain-openai (12:52:21)

נאכן קורצן איבערבליק גייט דער קורס טיפער אריין אין יעדע ליבראריע באזונדער.

ערשטנס LangChain, מיטן באפעל pip install langchain. פארוואס? ווייל LangChain איז דער הויפט-פריימווארק וואס העלפט בויען אפליקאציעס מיט גרויסע שפראך-מאדעלן (LLM), און עס גיט די יסוד-געצייג צו שאפן סטרוקטורירטע LLM ארבעטס-פלוסן. אנשטאט גלייך רופן א מאדעל און אליין שרייבן

אלעס מיט דער האנט, גיט LangChain פארטיגע טיילן: פראמפטס, chains, אגענטן, זכרון (memory) און דאקומענט-לאדערס (document loaders).

דערנאך: pip install langchain-openai. דער טעם: LangChain אליין אנטהאלט נישט די מאדעל-פארזארגער. עס פארבינדט זיך מיט פארשידענע פארזארגער דורך באזונדערע אינטעגראציע-ליבראריס. langchain-openai איז די בריק צווישן LangChain און די OpenAI מאדעלן, און זי לאזט באניצן מאדעלן ווי GPT 5.2, GPT-4o אדער GPT 5.4.

די OpenAI SDK און דער פלוס פון א בקשה (12:53:21)

דער נעקסטער: pip install openai. דאס איז דער אפיציעלער OpenAI Python SDK. ער באהאנדלט די API קאמוניקאציע, די אויטענטיפיקאציע, דאס שיקן פראמפטס און דאס באקומען די ענטפערס פונעם מאדעל. LangChain באניצט אינעווייניג די דאזיגע ליבראריע כדי צו רעדן מיט די OpenAI סערווערס.

דער קורס ערקלערט דעם פלוס אזוי: מען שרייבט קאוד; פון דארט גייט עס צו LangChain; פון LangChain צו דער OpenAI ליבראריע; פון דארט צום OpenAI API סערווער; און דערנאך קומט צוריק דער ענטפער פונעם מאדעל. אט דאס איז וואס עס טוט זיך פונקטליך אין דער פראגראם וועו מען באניצט LangChain.

python-dotenv, ChromaDB און FAISS (12:54:21)

דערנאך ווערט באשריבן די ליבראריע python-dotenv. זי העלפט פירן סביבה-פאריאבלען (environment variables), ספעציעל סודות ווי API שליסלען. אנשטאט שרייבן דעם API שליסל גלייך אינעם קאוד, היט מען אים אפ אין א .env טעקע. אין דער לעקציע ווערט אבער באמערקט אז דאס איז איין וועג; אינעם דעמא איז דער OpenAI API שליסל "האַרדקאָודעד" און ווערט געלאדן מיט דער OS ליבראריע וואס איז דא אין Python.

דערנאך Chroma. ChromaDB איז א וועקטאר-דאטאבאזע וואס היט אפ embeddings. embeddings זענען נומעריישע פארשטעלונגען פון טעקסט, וואס ווערן באניצט פאר סעמאנטישן זוכן (זוכן לויטן באדייט). א ביישפיל: אויב מען בויט א סיסטעם וואו באניצער פרעגן וועגן PDF טעקעס, העלפט ChromaDB מיט דריי זאכן: עס היט אפ די embeddings פון די דאקומענטן, עס זוכט ענליכע טעקסט-שטיקלעך (chunks), און עס שיקט דעם פאסיגן קאנטעקסט צום LLM. די טעכניק הייסט Retrieval Augmented Generation , אדער RAG.

דערנאך FAISS. pip install faiss-cpu איז נאך א וועקטאר-זוך מאשין, געשאפן דורך דער פירמע Meta. עס טוט זייער שנעלע ענליכקייט-זוכן צווישן embeddings, און ווערט אפט באניצט ביים בויען דאקומענט-זוך סיסטעמען, AI געהילפן און רעקאמענדאציע-מאשינען. ביי גרויסע דאטא-סעטס איז עס שנעלער ווי אסאך אנדערע וועקטאר-זוך געצייג.

tiktoken און די אלגעמיינע ארכיטעקטור (12:55:55)

דערנאך tiktoken, וואס ווערט באצייכנט אלס א זייער וויכטיגע ליבראריע. דער LLM פראצעסירט טעקסט אין טאקענס, נישט אין ווערטער. tiktoken העלפט ציילן טאקענס און פירן די טאקען-גרענעצן. פארוואס איז דאס וויכטיג? ווייל LLMס האבן א מאקסימום צאל טאקענס, און אז מען גייט אריבער די גרענעץ קומען ארויס טעותים (errors). ציילן טאקענס העלפט אפטימיזירן פראמפטס, און די ליבראריע העלפט מיט עפעקטיוון באהאנדלען פון פראמפטס.

דער קורס גיט דערנאך די פשוטע ארכיטעקטור פונעם גאנצן סעטאפ:

- מען שרייבט Python קאוד, למשל א טעקע app.py.
- דער קאוד גייט צום LangChain פריימווארק.
- LangChain באניצט די langchain-openai אינטעגראציע.
- די אינטעגראציע פארבינדט זיך מיטן OpenAI SDK.
- דער SDK רופט דעם API – למשל דעם מאדעל 5.2 – אויפן OpenAI מאדעל-סערווער.
- דער מאדעל גיט צוריק דעם ענטפער לויט דער פראגע וואס מען האט געפרעגט.

דערמיט ענדיגט זיך דער טייל פון אויפשטעלן LangChain.

באקומען א API שליסל און שרייבן דעם ערשטן פראגראם (12:56:59)

דער נעקסטער שריט: גיין צו דער OpenAI API פלאטפארמע און דארט נעמען א OpenAI API שליסל. אויב מען האט נאך נישט, שאפט מען איינעם – עס איז דא אן אפציע "Create new secret key". מען קאפירט דעם שליסל און מען ניצט אים אין פראגראם.

אין דער לעקציע ווערט געוויזן א פשוטער LangChain סעטאפ אין app.py:

- מען שטעלט אריין דעם OpenAI API שליסל (די OS ליבראריע ווערט אינעם מאמענט נישט באניצט).
- מען אימפארטירט ChatOpenAI פון langchain_openai.
- מען שאפט א פאריאבל llm מיט ChatOpenAI און גיט צוויי פאראמעטערס: model און temperature. די טעמפעראטור לאזט מען אויף 0 דערווייל, און דעם מאדעל שטעלט מען אויף 5.4.
- מען שאפט א פאריאבל response, וואו מען רופט invoke מיטן פראמפט "Explain LangChain in simple words".
- צום סוף דרוקט מען response.content – דאס הייסט, דעם אינהאלט וואס דער מאדעל גיט צוריק.

דער קערן פונעם קאוד זעט אויס בערך אזוי:

```
llm = ChatOpenAI(model="gpt-5.4", temperature=0)

response = llm.invoke("Explain LangChain in simple words")

print(response.content)
```

לויפן דעם פראגראם און דער רעזולטאט (12:58:31)

כדי צו לויפן דעם פראגראם, ווייל דער טעקע-נאמען איז app.py און די Python ווערסיע איז 3.14, ניצט מען python3 און דעם טעקע-נאמען:

```
python3 app.py
```

נאכן דריקן Enter הייבט עס אן ארבעטן לויטן פראמפט, און נאך א קורצער ווייל קומט דער ענטפער. דער מאדעל ערקלערט וואס LangChain איז: א געצייג-קאסטן (toolkit) צו בויען אפליקאציעס וואס באניצן LLMס ווי GPT. דערנאך ערקלערט עס אין פשוטע ווערטער מיט א משל: מען קען טראכטן פונעם LLM ווי א מוח, און LangChain איז די "וויירינג" (דראָטן-פארבינדונג) וואס העלפט דעם מוח רעדן מיט דאקומענטן,

באניצן דרויסנדיגע געצייג, אפהיטן דעם קאנטעקסט און דאס גלייכן. מען זעט אז מען האט מיט הצלחה באניצט LangChain מיט Python.

וואס זענען Chains (12:59:32)

דערנאך ערקלערט דער קורס דעם באגריף chains. אין LangChain איז א chain א סדר פון טריט, וואו דער רעזולטאט פון איין טריט ווערט דער אריינגאנג פונעם נעקסטן. אנשטאט שיקן איין פראמפט צום שפראך-מאדעל און באקומען איין ענטפער, לאזן chains בויען סטרוקטורירטע ארבעטס-פלוסן.

א משל: א "אסעמבלי ליין" אין א פאבריק, וואו יעדע סטאנציע טוט א באשטימטע ארבעט און דער רעזולטאט גייט ווייטער צו דער נעקסטער סטאנציע. אזוי אויך אין LangChain טוט יעדער טריט א געוויסע ארבעט – פארמאטירן א פראמפט, רופן דעם מאדעל, ארויסנעמען דאקומענטן, אדער באארבעטן דעם רעזולטאט.

א ביישפיל: א סיסטעם וואס מאכט א קיצור פון א דאקומענט. דער פלוס:

- דער באניצער גיט א אריינגאנג.
- מען לאדט דעם דאקומענט.
- מען צעטיילט דעם דאקומענט.
- מען שיקט די שטיקלעך (chunks) צום מאדעל.
- מען שאפט דעם קיצור.
- מען באקומט דעם ענדגילטיגן ענטפער.

יעדער פון די טריט איז א טייל פונעם chain, און LangChain פירט דעם פלוס צווישן זיי, אזוי אז די פראגראמירער דארפן נישט אליין קאארדינירן אלעס. chains זענען זייער נוצליך ווען די ארבעט פאדערט עטליכע באארבעטונגס-שטאפלען אנשטאט איין פראמפט.

פראמפטס און Prompt Templates (13:01:37)

דערנאך ווערט ערקלערט וואס א פראמפט איז: פשוט די אנווייזונג וואס מען גיט דעם גרויסן שפראך-מאדעל. עס זאגט דעם מאדעל וואס צו טון און ווי צו ענטפערן. למשל, "Explain machine learning in simple steps" איז א פשוטער פראמפט.

אבער אין אמתע אפליקאציעס זענען פראמפטס אפט דינאמיש – א טייל פונעם פראמפט ענדערט זיך לויטן באניצערס אריינגאנג. דא קומט צו נוץ א prompt template: א פראמפט וואס מען קען ניצן ווידער און ווידער, מיט א "פלאץ-האלטער" (placeholder).

א ביישפיל: אויב דער באניצער גיט topic = "vector databases", ווערט דער ענדגילטיגער פראמפט: "Explain the concept of vector databases in simple terms". מען דעקלארירט דעם ווערט פון topic, שטעלט אריין "vector databases", און דער פראמפט באקומט די פארעם. דאס מאכט די סיסטעם אסאך מער בייגעוודיג, ווייל דער זעלבער טעמפלעיט קען ווערן באניצט פאר פארשידענע אריינגאנגען. דערפאר זענען prompt templates זייער וויכטיג, און מען וועט זיי ניצן ווייטער אין קורס.

וואס איז RAG און זיין פלוס (13:03:12)

דערנאך גייט מען איבער צו RAG, און פארוואס עס איז אזוי שטארק ביים ארבעטן מיט LangChain. RAG לאזט אן אפליקאציע ענטפערן פראגעס מיט הילף פון דרויסנדיגע דאקומענטן. מען האט שוין געזעדט וועגן

וועקטאר-שטאָרס ווי FAISS אדער ChromaDB, וואס היטן אפ embeddings און נעמען ארויס די פאסיגע שטיקלעך דאטא – און דערפאר איז RAG אזוי וויכטיג.

דער פלוס פון RAG, ווי ער ווערט געוויזן:

- מען האט א מקור-דאקומענט, למשל data.txt.
- מען לאדט דעם דאקומענט מיט Python.
- מען צעטיילט אים אין קליינע טעקסט-שטיקלעך.
- מען שאפט embeddings פאר יעדן שטיקל. אן embedding איז נישט מער ווי א וועקטאר-פארעם פון אפהיטן טעקסט – אין נומערן.
- מען היט אפ די embeddings אין א וועקטאר-דאטאבאזע ווי FAISS אדער Chroma.
- ווען דער באניצער פרעגט א פראגע, למשל "What is LangChain?", שאפט מען אן embedding פאר דער פראגע.
- מען טוט א ענליכקייט-זוכן אין דער וועקטאר-דאטאבאזע.
- מען נעמט ארויס די בעסט-פאסנדע שטיקלעך וואס דערציילן וואס LangChain איז.
- מען בויט דעם ענדגילטיגן פראמפט: קאנטעקסט פלוס פראגע, און מען גיט עס צום LLM, למשל GPT 5.4.
- דער ענטפער ווערט געוויזן.

אזוי זעט אויס דער Retrieval Augmented Generation צוגאנג ווען מען בויט אים מיט LangChain.

א RAG-אפליקאציע מיט LangChain און FAISS (13:05:18)

וואס RAG טוט, און די ביבליאטעקן (13:05:18)

דער קורס הייבט אן מיט א קורצן סך-הכל: Retrieval Augmented Generation (RAG) האט א זייער שטארקע אנווענדונג, ווייל עס גיט קאנטעקסט צו דער פראגע וואס מען שטעלט דעם LLM. דער סיסטעם זוכט אין אינדרויסנדיגע דאקומענטן אדער אין אן אינדרויסנדיגע דאטא-באזע, און געפינט דעם קאנטעקסט פון דעם וואס דער באניצער וויל טון. נאכדעם ווערט געוויזן א פראקטישע דעמאנסטראציע, כדי מען זאל עס בעסער פארשטיין.

פאר דער דעמאנסטראציע דארף מען עטליכע ביבליאטעקן. רוב זענען שוין פריער אינסטאלירט געווארן: langchain און langchain-openai. מען קען נאך אינסטאלירן langchain-core, openai, faiss-cpu און tiktoken. דערצו ווערט אויך אינסטאלירט די langchain-community ווערסיע. עס ווערט באמערקט אז די אינסטאלאציע קען נעמען א ביסל צייט.

די דאטא-טעקע און די אימפארטן אין app.py (13:06:20)

ערשט שאפט מען א נייע טעקע מיטן נאמען data.txt. אין איר שטייט א טעקסט וואס גיט א געדאנק ווען RAG און LangChain. דאס איז א וויכטיגע טעקע, ווייל זי ווען דינען אלס דער קאנטעקסט פאר די ענטפערס.

דערנאך שרייבט מען די הויפט-טעקע app.py. די אימפארטן זענען:

- os: וויכטיג ווען מען לייגט אריין דעם OpenAI API שליסל.

- פון langchain_openai: ChatOpenAI און OpenAIEmbeddings.
- פון langchain_community.document_loaders: TextLoader.
- פון langchain_text_splitters: CharacterTextSplitter.
- פון langchain_community.vectorstores: FAISS, די וועקטאר-דאטא-באזע פון Meta.
- פון langchain_core: ChatPromptTemplate.
- פון langchain_core.output_parsers: StrOutputParser, וואס פארסט דעם רעזולטאט.

אריינלאדן דעם דאקומענט (13:07:52)

דער ערשטער שריט איז אריינשטעלן דעם OpenAI API שליסל דורך os.environ.

דער צווייטער שריט איז לאדן דעם טעקסט-דאקומענט. מען שאפט א ווערייעבל loader מיט TextLoader. ער לאדט די טעקע data.txt מיטן ענקאדינג UTF-8. דערנאך שאפט מען א ווערייעבל documents און מען רופט loader.load(), וואס לאדט דעם דאקומענט פון דעם פאלדער.

צעטיילן אין שטיקלעך (chunks) און די באדייט פון overlap (13:08:23)

דער נעקסטער שריט איז צעטיילן דעם דאקומענט אין שטיקלעך (chunks), כדי מען זאל זיי קענען פארוואנדלען אין embeddings. מען שאפט א ווערייעבל text_splitter מיט CharacterTextSplitter און גיט אים צוויי פאראמעטערס: chunk_size=500 און chunk_overlap=50.

כדי צו ערקלערן וואס overlap מיינט, ווערט געבראכט א ביישפיל מיט אזא פאראגראף: "LangChain is a framework which is used to build applications powered by large language models, and it allows developers to connect LLMs with tools, memory and external data sources."

- אויב מען שטעלט chunk_size אויף 50 און מען גיט קיין overlap נישט, ווערט chunk 1: "LangChain is a framework which is used to build applications", און chunk 2: "powered by large language models, it allows developers". די צוויי שטיקלעך האבן גארנישט געמיינואם.
- מיט overlap ווערט chunk 1: "LangChain is a framework which is used to build applications", און chunk 2 הייבט אן מיט "to build applications powered by large language models". אזוי זעט מען אז א טייל טעקסט חזרט זיך איבער צווישן ביידע שטיקלעך.

אין app.py איז chunk_overlap געשטעלט אויף 50, האלט ווען זיין אזא איבערשניידונג. צום סוף שאפט מען א ווערייעבל docs און מען רופט text_splitter.split_documents(documents).

embeddings, FAISS און דער retriever (13:10:28)

יעצט קומט א וויכטיגער שריט. מען ניצט די פונקציע OpenAIEmbeddings צו שאפן embeddings און מען לייגט זיי אריין אין א ווערייעבל embeddings. דערנאך לייגט מען די שטיקלעך אריין אין דער FAISS וועקטאר-דאטא-באזע:

```
FAISS.from_documents(docs, embeddings) = vectorstore
```

צום סוף שאפט מען א retriever:

```
vectorstore.as_retriever(search_kwargs={"k": 3}) = retriever
```

עס ווערט ערקלערט וואס דא קומט פאר: די שטיקלעך ווערן פארוואנדלט אין embeddings און ווערן געהאלטן אין FAISS. דערנאך מאכט מען פון דעם vector store א retriever, וואס זאל צוריקגעבן די דריי בעסטע רעזולטאטן פון דער וועקטאר-דאטא-באזע. אזוי פאראלעל מען דעם ענטפער לויט די דריי שטיקלעך וואס פאסן זיך דאס בעסטע צו דער פראגע.

דער LLM און דער פראמפט-מוסטער (13:12:02)

מען שאפט א ווערי‌עבל llm מיט ChatOpenAI פון langchain_openai. מען גיט אן דעם נאמען פונעם מאדעל, און מען שטעלט temperature אויף 0. לויט דער ערקלערונג איז א העכערע טעמפעראטור דא נישט נויטיג.

דערנאך שאפט מען א פראמפט-מוסטער (prompt template), וואס איז זייער וויכטיג. ווי פריער דערמאנט, הענגט מען אזא מוסטער צו צו יעדער אינפוט פונעם באניצער. אזוי לייגט מען בלויז אריין די טעמע און באקומט דעם פאסיגן רעזולטאט. דער טעקסט פונעם מוסטער זאגט בערך: "You are a helpful assistant. Use only the provided context to answer the question. If the answer is not in the context, say: I could not find that in the document." נאכדעם קומען צוויי פלעצער: {context} און {question}.

דער פראגע-ענטפער לופ און די chain (13:13:05)

מען הייבט אן א פראגע-ענטפער לופ. ערשט ווייזט מען אן א מעלדונג אז די RAG-אפליקאציע איז גרייט, און אז מען קען שרייבן exit כדי ארויסצוגיין. אין א while True לופ פרעגט מען דעם באניצער א פראגע מיט דער פונקציע input. אויב די פראגע, אין קליינע אותיות, איז "exit", דרוקט מען "Goodbye" און מען ברעכט ארויס פונעם לופ.

אנדערש זוכט מען די פאסיגע שטיקלעך: מען שאפט א ווערי‌עבל retrieved_docs און מען גיט די פראגע איבער צום retriever. דערנאך פארבינדט מען דעם קאנטעקסט מיט דער פונקציע join און א list comprehension, וואס נעמט דעם page_content פון יעדן doc אין retrieved_docs.

נאכדעם בויט מען די chain:

prompt | llm | StrOutputParser() = chain

דערנאך באקומט מען דעם ענטפער מיט chain.invoke, און מען גיט איבער ערשט דעם context און דערנאך די question. צום סוף דרוקט מען אויס דעם ענטפער (response).

לויפן די פראגראם און די רעזולטאטן (13:14:36)

מען לויפט די פראגראם מיטן באפעל python3 app.py, און עס ערשיינט "Ask a question".

- ערשטע פראגע: "Explain what is RAG". דער ענטפער: RAG שטייט פאר Retrieval Augmented Generation, און עס דערמעגלעכט אפליקאציעס צו ענטפערן פראגעס מיט הילף פון דעם צוגעשטעלטן קאנטעקסט.
- צווייטע פראגע: "Which model is better, GPT-4o mini or GPT-5.4?". ווייל אזא אינפארמאציע איז נישט פארהאן אין דעם דאקומענט, ענטפערט דער סיסטעם: "I could not find it".

דער קורס באטאנט אז דאס איז איינע פון די וואונדערליכע אנוועגונגען פון RAG. מען שפייזט דעם מאדעל מיט דאטא, ער זוכט דארט דעם קאנטעקסט, און ערשט דערנאך גיט ער דעם ענטפער.

חזרה: וואס זענען embeddings און FAISS (13:16:12)

דער קורס חזרט נאכאמאל איבער די שריטן פון embeddings ווייטער, ווייל זיי זענען זייער וויכטיג. OpenAIEmbeddings פארוואנדלט די טעקסט-שטיקלעך אין וועקטאר-embeddings, דאס הייסט, עס שטעלט פאר טעקסט אלס נומערן. למשל, דער זאץ "LangChain is a framework" קען ווערן געהאלטן אלס א וועקטאר ווי 0.23, 0.44, 0.91- און אזוי ווייטער. די וועקטארן כאפן אריין דעם באדייט פונעם טעקסט. דאס לאזט דעם סיסטעם זוכן לויט סעמאנטישער ענלעכקייט, און נישט לויט די גענויע ווערטער.

FAISS.from_documents לייגט די embeddings אריין אין FAISS. דער FAISS vector store דינט ווי א דאטא-באזע פאר אלע וועקטארן. דער פראצעס זעט אויס אזוי: א דאקומענט-שטיקל גייט צום embedding, און דער רעזולטאט ווערט געהאלטן אין FAISS. שפעטער, ווען א באניצער שטעלט א פראגע, גענפינט FAISS די שטיקלעך וואס זענען דאס מערסטע ענלעך. מיט אנדערע ווערטער, FAISS איז די זוך-מאשין פונעם RAG-סיסטעם.

דערנאך ווערט ערקלערט דער retriever: דאס איז דער קאמפאנענט וואס זוכט אין דער וועקטאר-דאטא-באזע. ווען מען שטעלט k=3, מיינט דאס אז דער retriever גיט צוריק די דריי שטיקלעך וואס זענען דאס מערסטע רעלעוואנט. למשל, ווען דער באניצער פרעגט "What is LangChain".

ענדיגן דעם RAG פראיעקט, אגענטן און וועקטארן (13:18:47)

דער LLM, טעמפעראטור און דער פראמפט-מוסטער (13:18:47)

דער retriever זוכט אין דער וועקטאר-דאטאבאזע און ברענגט צוריק די מערסטע פאסיגע שטיקלעך (chunks). דערנאך שאפט מען דעם גרויסן שפראך-מאדעל (LLM) דורך ChatOpenAI. מען נעמט דעם מאדעל 5.4 מיט temperature 0. דער קורס ערקלערט אז דאס איז איינער פון די וויכטיגסטע טריט, ווייל דא באשטימט מען דעם LLM וואס וועט שרייבן דעם לעצטן ענטפער. דער OpenAI מאדעל שאפט די ענטפערס.

עס ווערט איבערגעחזרט וואס temperature טוט: עס קאנטראלירט ווי צופעליג דער ענטפער איז. ביי 0 קריגט מען מער פאקטישע ענטפערס, ביי 1 מער שעפערישע. ביי RAG סיסטעמען האלט מען עס געוועלך נידעריג, כדי צו פארמינערן hallucinations, ד.ה. אז דער מאדעל זאל נישט אויסטראכטן זאכן.

דערנאך שאפט מען א prompt template. אין דעם פראמפט שרייבט מען למשל: "דו ביסט א הילפסאמער אסיסטענט, און אויב דער ענטפער איז נישט אין דעם קאנטעקסט, זאג אז דו האסט עס נישט געפונען אין דעם דאקומענט". אזוי איז געווען אין דעם אויסגאנג פריער, ווען מען האט געפרעגט "וועלכער מאדעל איז בעסער". דער פראמפט זאגט דעם מאדעל ווי זיך צו פירן. ער האט צוויי ליידיגע ערטער (placeholders):

- context – דא קומען אריין די דאקומענט-שטיקלעך וואס דער retriever האט געפונען.
- question – דא קומט אריין די פראגע פונעם באניצער.

דער פארטיגער פראמפט ווערט געשיקט צום מאדעל, כדי ער זאל פארשטיין דעם קאנטעקסט און ענטפערן.

א ביישפיל: דער קאנטעקסט קען זיין "LangChain is a framework which is used to build applications powered by large language models", און די פראגע קען זיין "What is LangChain?". אזוי פילט מען אן די צוויי ערטער.

די פראגע-לופ, דער retriever און דער context (13:20:52)

ביים בויען דעם פראגע-און-ענטפער טייל בעט מען צוערשט דעם באניצער אריינצושרייבן א פראגע. דאס לאזט דעם באניצער פרעגן איין פראגע נאך די אנדערע, און די לופ לויפט ווייטער ביז דער באניצער שרייבט exit.

דערנאך נעמט מען ארויס די פאסיגע שטיקלעך דורך דעם retriever. דער טריט זוכט אין דער וועקטאר-דאטאבאזע: די פראגע ווערט פארוואנדלט אין an embedding, און מען פארגלייכט עס מיט די געשפייכלערטע דאקומענט-וועקטארן. FAISS גיט צוריק די ענליכסטע שטיקלעך.

דערנאך שאפט מען א context. מען לייגט צוזאם אלע געפונענע שטיקלעך אין איין בלאק טעקסט, און דאס ווערט דער קאנטעקסט וואס מען שיקט צום LLM. דער ביישפיל פון פריער ווערט ווידער געגוצט: "LangChain is a framework used to build applications powered by large language models", ווי א ביישפיל פון א קאנטעקסט וואס מען האט ארויסגענומען.

די LangChain רער-ליניע (pipeline) און דער ענטפער (13:22:25)

שפעטער בויט מען א LangChain pipeline, אין דער פארעם:

prompt | llm | output_parser = chain

דער קורס ערקלערט: דער פראמפט גייט צום LLM, און פון דארט צום output parser. דער parser פארוואנדלט דעם ענטפער פונעם מאדעל אין פשוט'ן טעקסט.

צום סוף שאפט מען דעם ענטפער: מען מאכט א ווריאבל response, און מען רופט chain.invoke מיט דעם context און דער question. אזוי שיקט מען די ווערטן פאר די צוויי ערטער. דער context קומט פון די דאקומענט-שטיקלעך, די question קומט פונעם באניצער. LangChain לייגט די ווערטן אריין אינעם פראמפט, שיקט די לעצטע בקשה צום מאדעל, און מען דרוקט אויס דעם רעזולטאט. דער מאדעל שרייבט דעם ענטפער לויטן קאנטעקסט פונעם דאקומענט. דאס איז געווען א ביישפיל-פראיעקט: מען נעמט אינפארמאציע פון א דאקומענט און מען גיט עס צום מאדעל, כדי ער זאל ענטפערן לויט דעם קאנטעקסט.

איבערחזרונג: ווי דער גאנצער RAG פראצעס ארבעט (13:24:29)

דער קורס חוז'ט איבער דעם גאנצן פראצעס, ווייל ער איז זייער וויכטיג:

- מען קריגט די פראגע פונעם באניצער.
- די פראגע ווערט פארוואנדלט אין embeddings.
- מען זוכט אין דער וועקטאר-DB.
- מען נעמט ארויס די פאסיגע דאקומענט-שטיקלעך.
- מען לייגט די שטיקלעך אריין אינעם פראמפט.
- מען שיקט דעם פראמפט צום LLM.
- דער ענטפער ווערט געשאפן.

דאס אלעס איז געמאכט געווארן מיט LangChain. מען רעקאמענדירט צו שטודירן די אפיציעלע LangChain דאקומענטאציע און דארט ווייטער צו פארשן.

אגענטן אין LangChain (13:25:32)

דערנאך הייבט מען אן רעדן פון אגענטן. אן אגענט פאראייניגט שפראך-מאדעלן מיט געצייג (tools). מען קען אים פרעגן פראגעס, און ער טראכט איבער (reasoning) און ענטפערט דערלויט.

דער פלוס פון אן LLM אגענט:

- מען גיט אן input צום LLM מאדעל.
- עס גייט צום "מאוד" (mode), וואס האט דריי זאכן: action, observation און finish.
- ביי action גייט עס צו די tools און פראצעסירט די זאך.
- ביי observation קוקט עס אן דעם רעזולטאט, און עס גייט צוריק צום מאדעל.
- צום סוף גיט דער מאדעל ארויס דעם ענטפער.

מען ווייזט אז אין דער דאקומענטאציע שטייט א וויכטיגע באמערקונג: די create_agent מעטאד פון LangChain בויט א runtime וואס איז געבויט אויף א גראף, און אונטן נוצט עס LangGraph.

די הויפט-טיילן: דער ערשטער איז דער model , וואס איז דער מאטאר פונעם טראכטן. ביים מאדעל זענען דא static models: מען שטעלט אים איין מאל איין, מען שאפט דעם אגענט, און ער בלייבט די גאנצע צייט אן ענדערונג. ביים ביישפיל אימפארטירט מען create_agent פון langchain און מען דעקלארירט א ווערייאבל agent. עס זעט אויס ענליך צו JavaScript: מען שאפט אן אביעקט פון create_agent מיט שליסל-און-ווערט פארן. מען גיט דעם מאדעל-נאמען, למשל GPT-5 . מען קען אויך צוגעבן נאך פאראמעטערס, ווי temperature, max tokens און timeout. די סדר איז: מען שאפט דעם מאדעל מיט ChatOpenAI, וואס קומט פון langchain_openai, און מען שאפט דעם אגענט מיט create_agent, וואס קומט פון langchain. דאס איז א קורצער אנהייב ווי צו ארבעטן דערמיט.

וואס איז א וועקטאר? (13:28:08)

א נייער טייל הייבט אן. א וועקטאר איז א שיינער נאמען פאר א רשימה פון נומערן. די נומערן זענען אבער נישט צופעליג. זיי זענען ווי א קליינער דיגיטאלער פינגער-אפדרוק, וואס זאגט וואס א זאך איז.

ביישפיל: וועו א קאמפיוטער קוקט מיט AI אויף א בילד פון א קאץ, זעט ער נישט קיין פוטער אדער וואנצעס. ער פארוואנדלט די קאץ אין א רשימה נומערן, ווי עס ווערט געוויזן אויפן עקראן. א בילד פון א הונט קריגט אן אנדערע רשימה. די צוויי זענען נישט גלייך, אבער זיי זענען נאענט, ווייל קעץ און הינט זענען ענליך: ביידע זענען חיות, מיט פוטער, פיס און שיינע פנימער. דער AI כאפט דאס אלעס אויף און דריקט עס אויס אין נומערן.

דאס ארבעט אויך פאר ווערטער און קולות. דאס ווארט happy ווערט א רשימה נומערן. דאס ווארט joyful לאנדט נאענט דערצו, ווייל עס האט א ענליכן טייטש. דאס ווארט angry לאנדט ווייט, ווייל די געפילן זענען גאנץ אנדערש. אויפן עקראן זעט מען דעם אונטערשייד צווישן joyful און angry. דער כלל: א וועקטאר איז א ספעציעלע רשימה נומערן, וואס העלפט א קאמפיוטער פארשטיין וואס א זאך מיינט. ער איז ווי א "טייטש-קאוד", א סודותדיגער פינגער-אפדרוק פאר געפילן, געדאנקען אדער ווי א זאך זעט אויס.

דער embedding מאדעל, די "טייטש-מאשין" (13:29:42)

פון וואו קומען די נומערן? פון דעם embedding model, וואס ווערט אנגערופן "meaning machine". ער איז ווי א קלוגער איבערזעצער, וואס פארוואנדלט מענטשלעכע זאכן, ווי ווערטער, בילדער אדער מוזיק, אין קאמפיוטער-זאכן, ד.ה. נומערן.

ביישפיל: מען גיט אים א זאץ וועגן פארביקע יום-טובים אין אינדיע, און ער שאפט א וועקטאר. דערנאך גיט מען אים א צווייטן זאץ, וואס רעדט אויך פון א פרייליכער ליכטיגער פייערונג, און ער קריגט אן אנדערע רשימה נומערן. די צוויי וועקטארן זענען זייער נאענט, ווייל ביידע זאצן רעדן פון דעם זעלבן געדאנק. די מאשין קוקט נישט נאר אויף די פונקטליכע ווערטער, נאר אויף וואס מען האט געמיינט. מען קען איר אפילו געבן א בילד אדער א שטיקל גיטאר-מוזיק, און זי וועט עס פארוואנדלען אין א וועקטאר לויט ווי עס קלינגט אדער ווי עס פילט זיך.

אזוי העלפט דער מאדעל קאמפיוטערס פארשטיין די אומארדנטליכע, שעפערישע און געפילפולע וועלט פון מענטשן, דורך עס פארוואנדלען אין ריינע, ארדנטליכע נומער-קאודס. יעדע זאך מיט א טייטש, ווי א ליד, א בילד פון א שמייכל-פנים אדער א זאץ, ווערט א וועקטאר, וואס מען קען דערנאך שפייכלערן, זוכן און פארגלייכן.

פארוואס נישט א געווענליכע דאטאבאזע? (13:31:45)

די פראגע איז: פארוואס זאל מען נישט אריינלייגן די נומער-קאודס אין א געווענליכן spreadsheet אדער דאטאבאזע? פאר מען רעדט וועגן וועקטאר-דאטאבאזעס, רעדט מען פון טראדיציאנעלע דאטאבאזעס, וואס מען נוצט שוין צענדליגער יארן. זיי זענען ווי א ריזיגער Excel בויגן, אדער א גוט-ארגאניזירטער שאפע מיט טעקעס. זיי היטן אינפארמאציע אין רייען און קאלומנעס, למשל א קאלומנע פאר נאמען. אויב מען פרעגט "ווער איז 10 יאר אלט?" אדער "ווער וואוינט אין Delhi?", געפינט די דאטאבאזע תיכף דעם ענטפער, ווייל מען גיט איר פונקטליכע פאקטן צו פארגלייכן. דערויף זענען טראדיציאנעלע דאטאבאזעס גוט: פאקטן, נומערן, נעמען און דאטומס, זאכן וואס זענען קלאר און סטרוקטורירט. אבער וואס טוט מען אז מען פרעגט די זעלבע דאטאבאזע "ווייז מיר א בילד וואס פילט זיך פרייליך", אדער "געפין לידער וואס האבן דעם זעלבן..."? דא ענדיגט זיך דער אפשניט.

וועקטאר-דאטאבאזעס: באדייטונג אנשטאט פונקט-ווערטער (13:32:46)

פארוואס געווענליכע דאטאבאזעס קענען נישט זוכן לויט באדייטונג (13:32:46)

דער קורס הייבט אן מיט א פראבלעם: א געווענליכע דאטאבאזע איז נישט געבויט צו פארשטיין געפילן אדער א "שטימונג". עס ווערט פארגליכן צו פרובירן געפינען א חלום מיט א קעלקולעיטאר. עס ארבעט פשוט נישט.

דער טעם איז ווייל טראדיציאנעלע דאטאבאזעס דארפן געניוע גלייכקייטן (exact matches). זיי פארשטייען נישט וואס זאכן מיינען אדער ווי זיי "פילן זיך".

א ביישפיל: אז מען זוכט דאס ווארט dog, ווען א נארמאלע דאטאבאזע ווייזן בלויז רעקארדס וואו עס שטייט פונקט דאס ווארט dog. אבער אויב אין א פייל שטייט puppy אדער golden retriever, ווען די דאטאבאזע נישט ווייסן אז זיי זענען פארבונדן, סיידן מען האט עס פריער אויסדריקליך אנגעזאגט. דאס איז א גרויסע באגרענעצונג.

דא קומען אריין וועקטאר-דאטאבאזעס. זיי זוכן נישט געוויע ווערטער. זיי זוכן באדייטונג, קאנטעקסט און ענליכקייט, און זיי געפינען זאכן וואס זענען נאענט אין באדייטונג, אפילו וועו דאס ווארט איז נישט דאסזעלבע. דערפאר ארבעטן אפליקאציעס ווי Spotify און Netflix אזוי גוט: זיי פארגלייכן נישט בלויז שליסל-ווערטער, נאר זיי "פארשטייען" דעם באניצער.

וואס געפינט זיך אינעווייניג אין א וועקטאר-דאטאבאזע (13:33:47)

א וועקטאר-דאטאבאזע באהאנדלט דאטא אויף א גאנץ אנדערן אופן. אנשטאט שורות און קאלאמען דארף מען זיך פארשטעלן א ריזיגע דריי-דימענסיאנאלע "גאלאקסיע" פון שיינענדיגע פינטעלעך. יעדער זאץ, ליד אדער בילד ווערט פארוואנדלט אין א וועקטאר, א רייע פון נומערן, און ווערט אזוי א פינטל אין דעם רוים.

פינטעלעך וואס זענען ענליך ליגן נאטירליך נאענט איינער צום אנדערן: קעץ נעבן הינט, פרייליכע ליידער נעבן פרייליכע געזאנגען, און אזוי ווייטער. דער קורס פארגלייכט עס צו א ריזיגן שפיל-פלאץ: קעץ און הינט טיילן די זעלבע רוטשבארע, בשעת ראקעטן און פלאנעטן האלטן זיך נעבן דעם טעלעסקאפ.

וואס פאסירט ביים זוכן: וועו מען שרייבט cute fluffy pet, פארוואנדלט דער קאמפיוטער דעם זאץ אין זיין אייגן פינטל. דאס נייע פינטל פאלט אריין אין דער גאלאקסיע, און די דאטאבאזע קוקט גלייך וועלכע פינטעלעך זענען דאס נאענטסטע און באלויכטן זיי: קעצלעך, הינטלעך, אפשר א בילד פון א קליין קינעלע. מען דארף נישט קיין געוויע ווערטער. אלעס הענגט אפ אין דער ווייטקייט אינעם "רוים פון באדייטונג".

די שנעלקייט-טריקס: ANN (13:34:47)

דער קורס שטעלט א פראגע: עס רעדט זיך פון מיליאנען און אפילו ביליאנען פינטעלעך אין דער גאלאקסיע. איז דען נישט זייער פאמעלעך צו זוכן דאס נאענטסטע? ווי אזוי געפינט "Robo", דער פיגור וואס דער קורס ניצט אלס ביישפיל, די נאענטסטע פינטעלעך פאר מען קען נאך אפילו מאכן מיט די אויגן?

דער סוד: וועקטאר-דאטאבאזעס ניצן קלוגע קורצוועגן, מעטאדן ווי approximate nearest neighbor search, אין קורצן ANN. די טריקס לאזן Robo דורכלויפן דעם רוים און געפינען נאענטע גלייכקייטן, אן קאנטראלירן יעדעס פינטל באזונדער. די דריי הויפט-מכשירים:

- HNSW (Hierarchical Navigable Small World): מען זאל זיך פארשטעלן אז מען בויט שאסיי-ראמפס אין רוים. אנשטאט שפרינגען פון פינטל צו פינטל ווי אויף שטיינער אין א טייך, שפרינגט Robo איבער ריזיגע שטחים לויט פריער-געבויטע וועגן. עס איז ווי צו ניצן אן עקספרעס-עלעוועיטאר אין א גרויסן געביידע: עס ברענגט אים גלייך צו דער ריכטיגער געגנט.
- LSH (Locality Sensitive Hashing): Robo ווארפט אלע ענליכע פינטעלעך אריין אין דעם זעלבן עמער, ווי מען סארטירט מארמלען לויט קאליר. אז מען זוכט א בלויע מארמל, זוכט מען נישט דורך דעם גאנצן זעקל, נאר מען גייט גלייך צום בלויען עמער. LSH גרופירט ענליכע זאכן, און דאס זוכן ווערט אסאך שנעלער.
- PQ (Product Quantization): PQ פארקלענערט יעדן קאוד, ווי מען מאכט פון א גרויסן פייל א קליינעם zip-פייל. אנשטאט פארגלייכן לאנגע, דעטאלירטע קאודס, פארגלייכט Robo קליינע "פאזל-שטיקלעך". עס איז נישט הונדערט פראצענט גענוי, אבער עס איז זייער שנעל און געווענליך גענוג נאענט צו געפינען דאס וואס מען דארף.

דאס הייסט: Robo פליט דורך דעם רוים מיט עמערס, שאסייען און קליינע פאזל-קאודס, און געפינט ענטפערס אין מיליסעקונדעס, אפילו ביי ביליאנען זאכן. אבער זוכן איז בלויז איין טייל פון דער ארבעט.

וועקטאר-אינדעקס קעגן וועקטאר-דאטאבאזע (13:37:20)

עס ווערט באטאנט אז נישט אלעס וואס טוט וועקטאר-זוכן איז א פולע וועקטאר-דאטאבאזע. עס איז דא א גרויסער חילוק צווישן א וועקטאר-אינדעקס און א וועקטאר-דאטאבאזע. דער קורס פארגלייכט עס צו א טעלעפאן-בוך קעגן א ריכטיגע ביבליאטעק: ביידע האלטן זאכן, אבער איינס האט "סופער-כוחות".

א וועקטאר-אינדעקס איז ווי אן אלטמאדישער טעלעפאן-בוך: בלויז א ליסטע פון נעמען און נומערן. עס איז שנעל און נוצליך אויב מען וויל נאר שנעל אויפזוכן עפעס. אויב מען האט א סך vector embeddings, דאס הייסט די נומערן-ליסטעס פול מיט באדייטונג, קען א געצייג ווי FAISS זיי שנעל דורכזוכן און זאגן וועלכע זענען דאס נאענטסטע צו דעם אריינגאב.

אבער דארט ענדיגט זיך עס:

- מען קען נישט גרינג צולייגן נייע דאטא נאכדעם וואס דער אינדעקס איז געבויט.
- מען קען נישט דערהיינטיגן עקזיסטירנדע רעקארדס ווען עפעס ענדערט זיך.
- מען קען נישט צוטשעפען עקסטערע לעיבלס אדער קאטעגאריעס.
- קיין זיכערהייט, קיין מאניטארינג, קיין באקאפס.

עס איז ווי א שנעלע זוך-מאשין אן קיין וועג צו ארגאניזירן אדער פירן די דאטא אויף לאנגע צייט. דאס איז גוט פאר עקספערימענטן אדער קליינע אינערליכע געצייג, אבער נישט פאר אפליקאציעס אין דער ריכטיגער וועלט וואס ענדערן זיך כסדר.

וואס א פולע וועקטאר-דאטאבאזע קען טון (13:38:54)

א פולע וועקטאר-דאטאבאזע, ווי Pinecone אדער SingleStore, איז נישט בלויז א טעלעפאן-בוך. עס איז ווי א מאדערנע עפנטליכע ביבליאטעק, מיט קאמפיוטערס, שטאב, זיכערהייט און א גאנצע סיסטעם צו פירן אלעס. איר מעגליכקייטן:

- צולייגן, דערהיינטיגן און אויסמעקן: מען קען צולייגן נייע וועקטאר-דאטא ווי זי קומט אן, למשל נייע לידער, בילדער אדער דאקומענטן. מען קען דערהיינטיגן, למשל ווען א פראדוקט-ליסטינג ענדערט זיך אדער אן ארטיקל ווערט רעדאקטירט. און וואס מען דארף נישט מער קען מען אויסמעקן. דאס אלעס געשעט אין רעאלער צייט, אן דארפן איבערבויען אלעס פון אנפאנג.
- Metadata-פילטערן: לאמיר זאגן מען האט א מיליאן לידער אין דער דאטאבאזע. מען וויל נישט נאר לידער וואס "פילן זיך" ענליך, נאר אפשר לידער אויף Hindi, געשפילט מיט א גיטאר, וואס זענען פרייליך און באליבט ביי קינדער. יעדער וועקטאר קען טראגן טעגס אדער לעיבלס ווי זשאנער, שפראך, שטימונג אדער עלטער-גרופע. דאס לאזט פילטערן די רעזולטאטן צו זיין זייער ספעציפיש, נישט בלויז ענליך אין באדייטונג.
- זיכערהייט און צוטריט-קאנטראל: ווען מען האנדלט מיט סענסיטיווע אינפארמאציע, ווי באניצער-אויפפירונג, פריוואטע נאטיצן אדער פערזענליכע בילדער, מוז מען שיצן די דאטא. א פולע וועקטאר-דאטאבאזע האט באניצער-ראלעס און דערלויבענישן; multi-tenancy, דאס הייסט אפגעזונדערטע ערטער פאר יעדן באניצער אדער גרופע; און audit logs צו פארפאלגן ווער האט צוגעגריפן צו וואס און ווען. דאס זענען מוז-זאכן אין production, ספעציעל פאר אפליקאציעס מיט ריכטיגע באניצער.

• באקאפס און מאניטארינג: אז א טעלעפאן קראכט און מען פארלירט אלעס איז שוין שלעכט; מען זאל זיך פארשטעלן אז דאס פאסירט מיט אן אפליקאציע וואס מיליאנען מענטשן ניצן. דערפאר זענען באקאפס קריטיש. א וועקטאר-דאטאבאזע קען אויטאמאטיש מאכן באקאפס, ווייזן dashboards וואס פארפאלגן פערפארמענס, טעותים און געזונט, און העלפן זיך שנעל ערהוילן ווען עפעס גייט קאליע. דאס באקומט מען נישט ביי א פשוטן אינדעקס ווי FAISS.

• אינטעגראציע מיט דער סביבה: וועקטאר-דאטאבאזעס זענען געבויט צו ארבעטן גוט מיט די אנדערע געצייג, ציי עס איז LangChain אדער LlamaIndex פאר בויען AI-אגענטן, Tableau אדער Grafana פאר וויזועלע dashboards, אדער Python אדער Node.js פאר אפליקאציעס. זיי קומען מיט APIs און SDKs וואס פראגראמירער קענען גרינג אריינשטעקן. דאס מאכט זיי פאר דעם מאטאר הינטער אסאך מאדערנע AI-אפליקאציעס.

דאס גרויסע בילד: א וועקטאר-אינדעקס ווי FAISS איז נאר פאר זוכן, זייער שנעל אבער באגרענעצט. א וועקטאר-דאטאבאזע איז א פולער פעקל: זוכן, אויפהאלטן, פירן, שיצן און מאניטארן. עס איז ווי דער חילוק צווישן א קעלקולעיטאר און א גאנצע חשבונות-סיסטעם. וועט עס בויט א ריכטיגע AI-אפליקאציע פאר ריכטיגע מענטשן, דארף ניצן א וועקטאר-דאטאבאזע.

וואו מען ניצט שוין וועקטאר-דאטאבאזעס יעדן טאג (13:41:56)

מען דארף נישט זיין קיין AI-אינזשעניר צו ניצן וועקטאר-דאטאבאזעס; מען ניצט זיי שוין יעדן טאג:

• Spotify: וועט עס שלאגט פאר א ליד וואס פאסט פונקט צום שטימונג, איז דאס נישט קיין כישוף, נאר וועקטארן. יעדעס ליד וועט פארוואנדלט אין א וועקטאר, א סארט "מוזיק-פינגער-אפדרוק", און Spotify געפינט אנדערע לידער מיט ענליכן ריטם, שטימונג אדער וועטער.

• Netflix: יעדער פילם וועט א וועקטאר לויט זשאנער, טעמע און טעם, און דערנאך געפינט מען "שכנים". אז מען קוקט איין טרילער, ברענגט עס אנדערע וואס זענען ענליך, אפילו אין אן אנדערער שפראך.

• AI-ציטבאטס: וועט מען פרעגט א פראגע, לייענט עס נישט דורך יעדעס וואָרט. עס פארוואנדלט די פראגע אין א וועקטאר און זוכט די נאענטסטע באדייטונגען פון אלעס וואס עס האט געזען. דערפאר ענטפערט עס אזוי שנעל.

• באנקען: זיי ניצן וועקטאר-זוכן צו כאפן שווינדל. יעדע צאלונג וועט א וועקטאר. אויב איינע זעט אויס מאדנע, למשל זי קומט פון א לאנד וואו מען איז קיינמאל נישט געווען, וועט זי אנגעצייכנט. אזוי שיצן וועקטאר-דאטאבאזעס אויך דאס גאלט.

אן עקספערימענט מיט איין וואָרט: India (13:42:57)

אין דער לעקציע וועט באשריבן א ריכטיגער עקספערימענט: מען האט גענומען 10,000 נייעס-קעפלעך און זיי אריינגעשטופט אין א וועקטאר-דאטאבאזע מיטן נאמען SingleStoreDB. יעדעס קעפל איז פארוואנדלט געווארן אין א וועקטאר דורך דער "באדייטונג-מאשין" (דעם embedding-מאדעל) וואס מען האט פריער באשריבן.

דערנאך האט מען געזוכט מיט בלויז איין וואָרט: India. עס זענען ארויסגעקומען נייעס וועגן אינדיע'ס טעמעס, קולטורעלע אנגעלעגנהייטן, פאליטישע נייעס, און אפילו געשיכטעס וואס האבן בכלל נישט גענוצט דאס וואָרט India. דאס איז דער כח פון semantic search: זוכן לויט באדייטונג, נישט בלויז גלייכן אותיות.

איבערחזרונג (13:43:30)

דער קורס חזר'ט איבער די הויפט-פונקטן:

- וועקטארן זענען נומערן-"פינגער-אפדרוקן" וואס כאפן די באדייטונג.
- דער embedding-מאדעל איז די מאשין וואס פארוואנדלט זאכן אין וועקטארן.
- געוועלטיגט דאטאבאזעס פארשטייען נישט געפילן אדער שטימונגען.
- א וועקטאר-דאטאבאזע לייגט אויס די וועקטארן ווי שטערן אין רוים און געפינט די נאענטסטע שכנים.
- זי ניצט שנעלקייט-טריקס ווי HNSW, LSH און PQ.
- א פולע דאטאבאזע, נישט בלויז א זוך-אינדעקס, לאזט צולייגן, דערהיינטיגן, שיצן און פילטערן די דאטא.
- מען ניצט דאס אלעס שוין יעדן טאג.

דער קורס ענדיגט מיט א פראגע: וואס וועט מען בויען מיט דעם? למשל אן אפליקאציע וואס געפינט דעם פאסיגן GIF לויט ווי מען פילט זיך, אדער א פילם-געפינער לויט שטימונג.

א נייע לעקציע: פארוואס טראדיציאנעלע דאטאבאזעס זענען נישט גענוג (13:44:31)

דא הייבט זיך אן א נייע לעקציע וועגן דער זעלבער טעמע. זי עפנט מיט א פראגע: ווי אזוי זעען AI -געשייג ווי ChatGPT, Netflix-פארשלאגן, Spotify-פלייליסטס אדער מאדערנע ציטבאטס אויס צו פארשטיין פונקט וואס מען זוכט, אפילו ווען מען ניצט נישט די געניוע ווערטער? דער ענטפער: וועקטאר-דאטאבאזעס. די לעקציע וועט ערקלערן וואס א וועקטאר-דאטאבאזע איז, פארוואס זי איז געווארן איינע פון די וויכטיגסטע טעכנאלאגיעס אין AI, און ווי זי טרייבט מאדערנע אפליקאציעס ווי RAG-סיסטעמען, semantic search, AI-געהילפן, פארשלאג-מאשינען און נאך.

דער ביישפיל: מען בויט אן AI-ציטבאט פאר א פירמע מיט טויזנטער דאקומענטן: האנטביכער פאר ארבעטער, HR-פאליסיס, טריינירונג-אנווייזונגען, פראדוקט-מאנואלן, דאקומענטן פאר קאסטומער-סופּפּארט, א.א.וו. אן ארבעטער פרעגט: "Can I take a week off during December?" אין דעם HR-דאקומענט שטייט אבער אפשר: "employees are entitled to 20 annual vacations".

דער פונקט: די פראגע פונעם באניצער האט קיינמאל נישט געזוצט די ווערטער week off. א טראדיציאנעלע דאטאבאזע זוכט געוועלך געניע שליסל-ווערטער. ווייל week off איז אנדערש ווי vacation days אדער leaves, קען די דאטאבאזע האבן שווערigקייטן צו געפינען דעם ענטפער, אדער זי וועט אים בכלל נישט געפינען. דא הייבט טראדיציאנעלע זוכן אן דורכצופאלן. מענטשן פארשטייען באדייטונג; קאמפיוטערס פארשטייען טראדיציאנעל ווערטער. וועקטאר-דאטאבאזעס העלפן קאמפיוטערס פארשטיין די באדייטונג.

וואס איז א וועקטאר (13:46:35)

איידער מען פארשטייט וועקטאר-דאטאבאזעס, דארף מען פארשטיין וועקטארן. די לעקציע בעט צו פארלאזן אויף א רגע די מאטעמאטיק און טראכטן ווי דער מוח ארבעט: ווען איינער זאגט דאס ווארט doctor, פארבינדט דער מוח אויטאמאטיש מיט שפיטאל, מעדיצין, פאציענטן, געזונט-פלעגע, באהאנדלונג און דיאגנאז. מען האלט נישט בלויז די אותיות D-O-C-T-O-R; מען האלט די באדייטונג הינטער דעם ווארט.

AI-מאדעלן ארבעטן ענליך. זיי פארוואנדלען טעקסט, בילדער, אודיא אדער אפילו וויידאס אין א נומעריש פארשטעלונג וואס הייסט א וועקטאר. א וועקטאר איז פשוט א ליסטע פון נומערן וואס פארשטעלט די

באדייטונג פון דער דאטא. למשל, דער זאץ "I love watching movies" קען ווערן א רייע נומערן, ווי 24.78, 1.56 און נאך, און א צווייטער זאץ, "I enjoy films", וועט אויך שאפן א וועקטאר.

וועקטאר-דאטאבאזעס, RAG און Chroma DB (13:47:37)

עמבעדינגס: ווי ענליכע באדייטונגען ווערן ענליכע וועקטארן (13:47:37)

דער קורס הייבט אן מיט א באמערקונג אז ווען צוויי זאצן זענען זייער ענליך, הגם די ווערטער זענען אנדערש און נאר די באדייטונג איז כמעט די זעלבע, ליגן זייערע וועקטארן נאענט איינער צום אנדערן. דער פראצעס פון איבערמאכן דאטא אין וועקטארן הייסט "embedding generation", און די נומעריש פארשטעלונג וואס קומט ארויס הייסט אן "embedding".

פארוואס שאפן ענליכע באדייטונגען ענליכע וועקטארן? עס ווערט ערקלערט מיט א משל: מען זאל זיך פארשטעלן א דריי-דימענסיאנאלע מאפע. יעדעס וואָרט, יעדער זאץ, בילד אדער דאקומענט ווערט געשטעלט ערגעץ אויף דער מאפע. זאכן מיט ענליכע באדייטונגען שטייען צוזאמען, און זאכן וואס האבן נישט קיין שייכות שטייען ווייט איינס פונעם אנדערן.

ביישפילן וואס ווערן געגעבן:

- "dog" און "puppy" (הונט און הינטעלע) וועלן זיין נאענט.
- "car" און "vehicle" (קאר און פארמיטל) וועלן אויך זיין נאענט.
- "pizza" און "pasta" קענען זיין רעלאטיוו נאענט.
- אבער "pizza" און "airplane" (עראפלאן) וועלן זיין ווייט אפגעטיילט.

דער AI מאדעל לערנט זיך די פארבינדונגען אויטאמאטיש בשעת'ן טריינירן. דאס דערמעגליכט מאשינען צו פארשטיין באגריפן אנשטאט נאר צופאסן ווערטער, און דאס איז דער יסוד פון וועקטאר-דאטאבאזעס.

וואס איז א וועקטאר-דאטאבאזע? (13:48:38)

די דעפיניציע: א וועקטאר-דאטאבאזע איז א ספעציעלע דאטאבאזע וואס איז געבויט צו האלטן און זוכן וועקטארן אויף אן עפעקטיוו אופן. אנשטאט זוכן געניוע שליסל-ווערטער, זוכט זי ענליכע באדייטונגען.

ווי עס ארבעט:

- וועו א באניצער פרעגט א פראגע, ווערט די פראגע איבערגעמאכט אין א וועקטאר.
- די וועקטאר-דאטאבאזע פארגלייכט אים מיט מיליאנען געהאלטענע וועקטארן.
- זי געפינט די וועקטארן וואס זענען די מערסטע ענליך.
- די צוגעהעריגע דאקומענטן ווערן צוריקגעגעבן.

דער פראצעס הייסט "similarity search" (ענליכקייט-זוכעניש). די דאטאבאזע זוכט נישט ווערטער וואס שטימען, נאר באדייטונגען וואס שטימען, און דאס איז דער הויפט חילוק.

א ביישפיל פונעם לעבן: אן אנליין קראם. א קונה זוכט "I need comfortable footwear for jogging" (איך דארף באקוועמע שיך פאר לויפן). די פראדוקט-באשרייבונג אבער זאגט "lightweight running shoes with extra cushioning" (לייכטע לויף-שיך מיט עקסטערע ווייכקייט). א געווענליכע זוכעניש קען פארפעלן

געוויסע פאסיגע פראדוקטן, ווייל די געגויע ווערטער שטימען נישט. א וועקטאר-דאטאבאזע אבער פארשטייט אז "footwear" איז ענליך צו "jogging footwear", וואס איז ענליך צו לויפן מיט באקווענמער ווייכקייט. דערפאר קען זי צוריקגעבן זייער פאסיגע פראדוקטן אפילו ווען די פארמולירונג איז אינגאנצן אנדערש, און דאס שאפט א סך בעסערע זוך-איבערלעבונג.

ווי וועקטאר-דאטאבאזעס טרייבן RAG (13:50:11)

איינע פון די מערסט פאפולערע AI ארכיטעקטורן היינט איז RAG, וואס שטייט פאר "Retrieval Augmented Generation". דער קורס ערקלערט עס מיט א פשוט'ן משל פון אן עקזאמען:

- סצענאר איינס, א "פארמאכטע ספר" עקזאמען: מען קען נאר ענטפערן לויט וואס מען געדענקט. ווייסט מען נישט דעם ענטפער, שטעקט מען. דאס איז ענליך צו א גרויסן שפראך-מאדעל (LLM) וואס ארבעט אליין: עס קען ענטפערן נאר מיט אינפארמאציע וואס עס האט זיך געלערנט בשעת'ן טריינירן.
- סצענאר צוויי, אן "אפענע ספר" עקזאמען: מען האט צוטריט צו א לערן-ספר. וועז א פראגע קומט, געפינט מען דעם פאסיגן קאפיטל, ליינט מען די אינפארמאציע, באניצט מען זיך מיט שכל, און מען שרייבט דעם פערפעקטן ענטפער.

דאס איז געגוי ווי RAG ארבעט: דער LLM שפילט די ראלע פונעם תלמיד, און די וועקטאר-דאטאבאזע שפילט די ראלע פונעם לערן-ספר. וועז א באניצער פרעגט א פראגע, ווערט די פראגע א וועקטאר, די וועקטאר-דאטאבאזע ברענגט ארויס פאסיגע אינפארמאציע, און די אינפארמאציע ווערט געשיקט צום LLM. דער LLM שאפט א מער געגויען ענטפער. דאס פארמינערט באדייטנד די "hallucinations" (אויסגעטראכטע ענטפערס) און פארבעסערט די פארלעסלעכקייט.

דער פולער RAG ארבעטס-גאנג (13:51:12)

עס ווערט צוזאמענגעפאסט אין ניַן טריט:

- טריט 1: מען זאמלט די דאקומענטן.
- טריט 2: מען מאכט די דאקומענטן איבער אין עמבעדינגס מיט אן עמבעדינג-מאדעל.
- טריט 3: מען האלט די עמבעדינגס אין א וועקטאר-דאטאבאזע.
- טריט 4: דער באניצער שיקט אריין א פראגע.
- טריט 5: מען מאכט איבער די פראגע אין אן עמבעדינג.
- טריט 6: מען טוט א similarity search.
- טריט 7: מען ברענגט ארויס די פאסיגע דאקומענטן.
- טריט 8: מען שיקט די דאקומענטן צוזאמען מיט דער פראגע פונעם באניצער צום LLM.
- טריט 9: מען שאפט דעם ענדגילטיגן ענטפער.

דער גאנצער ארויסברענג-פראצעס פאסירט אין בלויז עטליכע סעקונדעס.

Chroma DB און אנדערע באנוצונגען (13:51:42)

דערנאך ווערט פארגעשטעלט איינע פון די פאפולערע וועקטאר-דאטאבאזעס היינט: Chroma DB. זי איז פריינטלעך פאר אנהייבער, גרינג צו אינסטאלירן, און פאסיג פאר לערנען און בויען לאקאלע RAG פראיעקטן.

דער קורס באטאנט אז אסאך מענטשן מיינען אז וועקטאר-דאטאבאזעס ווערן געוונצט נאר אין AI chatbots, אבער דאס איז נישט אמת. זיי טרייבן א סך אפליקאציעס:

- Spotify רעקאמענדירט לידער דורכן פארגלייכן הער-מוסטערן.
- Netflix רעקאמענדירט פילמען לויט דער ענליכקייט צווישן באניצער און אינהאלט.
- בילד-זוכעניש: מען לאדנט ארויף א בילד פון א שטול, און די סיסטעם קען געפינען שטולן וואס זעען אויס ענליך, אפילו אן קיין טעקסט-באשרייבונגען.
- אויסגעפינען שווינדל (fraud detection): אומגעוועניליכע טראנזאקציעס ליגן אין וועקטאר-רוים ווייט פון די נארמאלע טראנזאקציע-מוסטערן, און דאס הילפט אידענטיפיצירן חשד'דיגע אקטיוויטעטן, און נאך א סך.

דער Chroma וועבזייטל און די דאקומענטאציע (13:52:44)

די דעמאנסטראציע הייבט אן מיט זוכן Chroma DB אין א בראוזער (דא Google). מען קען אריינטיפן "chromadb" אדער גיין גלייך צום וועבזייטל trychroma.com. דארט געפינט מען דאקומענטאציע, פארשונג-מאטעריאל, פרייזן א.א.וו. מען זעט אז א סך פירמעס ניצן Chroma, און ווייטער אראפ ווערט אנגעוויזן: 15 מיליאן אראפלאדונגען א חודש, נידעריגע לאטענסי (latency), ביז 10% ביליגער, און "no engineering ops". עס ווערט אויך דערמאנט sparse vector search, vector search און full-text search, מיט קאוד-ביישפילן דערנעבן, און אויך א command line interface.

אין דער לעקציע ווערט געוונצט די דאקומענטאציע, כדי צו פארשטיין ווי Chroma ארבעט אויף אן אנהייבער-פריינטלעכן אופן. די אריינפירונג ווייזט וואס Chroma DB גיט: דאקומענט-האלטונג, עמבעדינגס, וועקטאר-זוכעניש, full-text און regex זוכעניש, multimodal retrieval, metadata filtering א.א.וו. עס זענען פאראן א quick start און א server אפציע, און ביישפילן ווי agentic search און code search. עס ווערט דערמאנט די Apache 2.0 לייסענס, און אויך Chroma Cloud. דורכן קליקן "get started" זעט מען קאוד-אנווייזונגען: א באזונדערע טוטאריאל פאר Python, איינע פאר TypeScript, און אויך פאר Rust. מען קען אינסטאלירן מיט הילף פון AI אדער מאנועל.

אינסטאלירן מיט Anaconda און Jupyter Notebook (13:54:46)

איידער מען הייבט אן, דארף מען האבן Anaconda אינסטאלירט אויפ'ן קאמפיוטער, ווייל די דעמא ווערט געטון אין Jupyter Notebook. צו אינסטאלירן Anaconda, טיפט מען "Anaconda download" אין בראוזער, גייט מען צום ערשטן לינק, קליקט "get started", לאדנט אראפ, און דריקט "next, next, next" — א פשוטער פראצעס.

אין Jupyter Notebook עפנט מען א נייעם Python notebook . דער ערשטער טריט איז אינסטאלירן chromadb. מען קען קאפירן דעם באפעל אין Anaconda Prompt, אדער אריינלייגן אין Jupyter Notebookוער עס פילט זיך נישט באקוועם מיט command prompt):

```
pip install chromadb
```

ווייל עס איז שוין אינסטאלירט, קומט ארויס "requirement already satisfied"; א נייער באניצער וועט זען ווי עס אינסטאלירט זיך. אין Jupyter לויפט מען א צעל מיט Shift+Enter.

קליענט, קאלעקציע און דאקומענטן (13:56:50)

דערנאך אימפארטירט מען (import chromadb) Chroma DB. אויב די אינסטאלאציע איז געלונגען, וועט נישט ארויסקומען קיין טעות.

דערנאך שאפט מען א קליענט (chromadb.Client()). דער קליענט איז וויכטיג, ווייל מיט אים שאפט מען קאלעקציעס. קאלעקציעס זענען וואו מען האלט די עמבעדינגס, דאקומענטן און נאך metadata; זיי אינדעקסירן די עמבעדינגס און דאקומענטן און דערמעגליכן עפעקטיוול ארויסברענג און פילטרירן. מען שאפט א קאלעקציע מיט א נאמען; די דאקומענטאציע ניצט "my_collection", און מען קען געבן א נאמען לויט'ן אייגענעם געשמאק. אין דער לעקציע ווערט צוגעלייגט א הערה אינעם קאוד: דאס שאפט אן אינסטאנץ פון א קאלעקציע, וואס ערמעגליכט צו טון אפעראציעס אויף דער וועקטאר-דאטאבאזע און צו ארבעטן מיט די אונטערליגנדע דאטא. נאך Shift+Enter איז די קאלעקציע געשאפן.

דער נעקסטער טריט: צולייגן דאקומענטן. Chroma האלט דעם טעקסט און באהאנדלט אויטאמאטיש די עמבעדינגס און אינדעקסירונג; מען קען אויך אויסקלייבן אן אנדער עמבעדינג-מאדעל. מען מוז געבן אייניציגארטיגע string ids פאר די דאקומענטן. אינעם ביישפיל זענען דא צוויי דאקומענטן: "This is a document about pineapple" (אננאס) און "This is a document about oranges " (מאראנצן), מיט די ids "id1" און "id2". אין דער לעקציע ווערן צוגעלייגט הערות אז id1 איז פארן אננאס-דאקומענט און id2 פארן מאראנצן-דאקומענט, און מען נעמט אראפ אן איבעריגן אפשטאנד. Shift+Enter.

פרעגן די קאלעקציע און אונטערזוכן די רעזולטאטן (13:59:23)

דער גאנצער ציל פון שאפן א קאלעקציע איז צו קענען פרעגן (query) די דאטא. מען שיקט א ליסטע פראגעס, און Chroma גיט צוריק די n מערסט ענליכע רעזולטאטן. אינעם ביישפיל איז די פראגע "This is a query document about hawaii"; Chroma מאכט דערפון אן עמבעדינג אליין. מען קען באשטימען וויפיל רעזולטאטן (דא n_results=2), און דער קאוד דרוקט אויס די רעזולטאטן.

ביים אונטערזוכן (inspect) זעט מען אז די פראגע וועגן Hawaii איז סעמאנטיש מער ענליך צום דאקומענט וועגן אננאס. נאכן לויפן דעם קאוד קומט טאקע ארויס: "This is a document about pineapple" ערשט.

דער לעצטער טריט אין דער דאקומענטאציע איז אליין פרובירן: דארט ווערט געפרעגט מיט "This is a document about florida", מיט א פולן ביישפיל. מען קען קאפירן די באפעלן אין Jupyter Notebook, פרובירן אייגענע וועריאציעס פון פראגעס, און זען די רעזולטאטן.

Pinecone: וועקטאר-דאטאבעיס און סעמאנטישע זוך (14:00:57)

סוף פון Chroma און ווייטערדיגע מעגליכקייטן (14:00:57)

אין דער לעקציע ווערן ווייטער געוויזן די באפעלן פון Chroma, איינס נאך דעם אנדערן. מען קען אויך אליין אויסדרוקן (print) די רעזולטאטן, און דער גאנצער טייל דינט אלס א פשוטער וועגווייזער פאר אנהייבער צו פארשטיין א וועקטאר-דאטאבעיס.

עס ווערט ערקלערט אז אין די ווייטערדיגע טריט פונעם וועגווייזער באניצט מען זיך מיט Chroma's in-memory client, כדי עס זאל זיין פשוט. דער client לאזט אויף א Chroma סערווער אינעם זכרון, און דערפאר גייט פארלוירן אלע דאטא וואס מען לייגט אריין, וועז די פראגראם ענדיגט זיך.

אין דער דאקומענטאציע געפינט מען נאך פעולות מיט collections:

- פארוואלטן collections: צולייגן, דערהיינטיגן און אויסמעקן דאטא
- קאנפיגורירן collections
- זוכן אין די דאטא (querying)
- דער CLI, וואס לאזט פון דער command line אויפשטעלן א Chroma סערווער
- לויפן אין cloud, ארבעטן אלס open source, מיגראציע און פארריכטן פראבלעמען (troubleshooting)

Chroma האט אויך א GitHub בלאט, און דער לינק דערצו וועלן אריינגעלייגט אין דער באשרייבונג פונעם ווידעא.

וואס איז ?Pinecone טראדיציאנעלע דאטאבעיס קעגן סעמאנטישע זוך (14:01:51)

דער קורס ערקלערט אז Pinecone איז נישט קיין טראדיציאנעלע דאטאבעיס ווי MySQL אדער MongoDB. א טראדיציאנעלע דאטאבעיס איז גוט ווען מען זוכט א געוויע וועלט, למשל א באניצער-נאמען, אן ID, א נומער פון א באשטעלונג אדער דער נאמען פון א פראדוקט. מען זוכט, און מען באקומט דעם געוויען רעזולטאט.

אבער אסאך אפליקאציעס אין דער וועלט ארבעטן נישט נאר מיט געוויע וועלטער. א מאל זוכט א באניצער מיט אנדערע וועלטער, און ער ווארט פונדעסטוועגן דעם ריכטיגן רעזולטאט. דער ביישפיל וואס וועלט געבראכט: איינער זוכט "I want to become a data analyst", אבער דער טיטל פונעם קורס אין דער דאטאבעיס הייסט "Business Analytics with PowerBI and SQL". ביי א געוועלטער זוך לויט שליסל-וועלטער קען דער רעזולטאט נישט ארויסקומען, ווייל די וועלטער זענען נישט די זעלבע. ביי סעמאנטישע זוך פארשטייט די סיסטעם דעם טייטש פון דער זוך: זי פארשטייט אז "ווערן א data analyst" איז פארבונדן מיט אנאליטיק, ווי PowerBI, SQL און ריפארטס. דא איז דער פלאץ וווּ א דאטאבעיס ווי Pinecone איז זייער נוצלעך.

בקיצור: Pinecone איז א וועקטאר-דאטאבעיס. זי היט אפ דאטא אין דער פארעם פון וועקטארן, און זי העלפט געפינען ענליכע זאכן לויטן טייטש, נישט נאר לויט געוויע שליסל-וועלטער.

וואס זענען וועקטארן? (14:03:24)

א וועקטאר איז פשוט א רשימה פון נומערן וואס שטעלט פאר דעם טייטש פון עפעס. דער ביישפיל: דער זאץ "I want to learn Python for data science". א מאשין קען נישט פארשטיין דעם זאץ ווי א מענטש, דערפאר פארוואנדלט מען אים אין נומערן דורך אן embedding מאדעל. דער רעזולטאט קען אויסזען למשל אזוי: 0.12, 0.45, 0.89, 0.76, 0.33, 0.67, 0.10 א.א.וו.

יעדער נומער כאפט א פארבארגענעם טייל פונעם טייטש פונעם זאץ. איין נומער קען פארשטעלן דעם נושא, א צווייטער די כוונה, און א דריטער ווי ענליך דער זאץ איז צו data science, וכדומה. מען באשטימט נישט מיט דער האנט די טייטשן. דער embedding מאדעל לערנט זיי אליין אויס.

א צווייטער זאץ, למשל "best course to start a career in data science" אדער "career in data analytics", וועלן אויך פארוואנדלט אין ענליכע וועקטארן, ווייל דער טייטש איז נאענט צום ערשטן זאץ. וועלן א באניצער זוכט עפעס, פארוואנדלט מען זיין זוך-שאלה אין א וועקטאר, און מען בעט Pinecone: "געפין מיר די אפגעהיטענע וועקטארן וואס זענען די נאענטסטע צו דעם וועקטאר". דאס הייסט similarity search, זוכן לויט ענליכקייט.

ביישפילן פון אפליקאציעס (14:05:28)

א ביישפיל פון א ריכטיגער אפליקאציע: א סיסטעם וואס רעקאמענדירט קורסן, אדער א סעמאנטישע זוך. מען בויט א זוך-פונקציע פאר אן edtech פלאטפארמע. מען היט אפ די באשרייבונגען פון די קורסן אין Pinecone אלס וועקטארן. וען א לערנער זוכט "I want to get a job in cloud computing", קען Pinecone צוריקגעבן קורסן ווי AWS Solutions Architect, Azure Fundamentals, Cloud Computing Bootcamp און DevOps with AWS. אפילו וען די ענויע וועלכע "job in cloud computing" זענען נישטא אין די טיטלען, געפינט Pinecone פונדעסטוועגן פאסיגע רעזולטאטן, ווייל זי זוכט לויטן טייטש.

דער זעלבער געדאנק קען ווערן באנוצט אין נאך אפליקאציעס:

- זוך אין קונה-סערוויס: באניצער פרעגן שאלות אין נאטירלעכער שפראך און באקומען דעם ריכטיגן הילף-ארטיקל.
- e-commerce זוך: איינער זוכט "comfortable shoes for running" און באקומט די פאסיגע ספארט-שיך.
- צופאסן רעזומעס: א recruiter זוכט "Python developer with machine learning experience" און באקומט ענליכע פראפילן פון קאנדידאטן.
- זוך אין דאקומענטן: אן ארבעטער זוכט פאליסיס פון דער פירמע, ריפארטס אדער טרייינינג-מאטעריאל, און באקומט וואס ער דארף.

אויפשטעלן א Pinecone חשבון (14:06:59)

נאכדעם וואס מען האט פארשטאנען וועקטארן און סעמאנטישע זוך, בויט מען א פשוטע FastAPI אפליקאציע וואס פארבינדט זיך מיט Pinecone . די אפליקאציע שטעלט צו די פעולות פון דער וועקטאר-דאטאבעיס דורך APIs. די ארבעט ווערט געטון אין VS Code.

פריער דארף מען אויפשטעלן א חשבון אויף Pinecone, וואס איז א פשוטער פראצעס:

- גיין צום אפיציעלן וועבזייטל פון Pinecone און קליקן "Start Building".
- זיך איינשרייבן (log in). מען קען זיך איינשרייבן דורך Google, GitHub, LinkedIn וכדומה (OAuth).
- ביים ערשטן איינשרייבן באקומט מען אן API key. אונטער "Get Started" ווערט עס געוויזן. עס איז שוין דא א default API key, אדער מען קליקט אויף "+", גיט א נאמען, און מען שאפט א נייעם key.
- די קאנפיגוראציע אין דער דעמא באניצט זיך מיט AWS cloud אין דער מזרח-ראיאן פון אמעריקע.

די פראיעקט-טעקעס און דער .env טעקע (14:08:28)

מען שאפט א הויפט-פאלדער, למשל "Python demo". עס זענען דא דריי וויכטיגע טעקעס: main.py, .env און א טעקע פאר Pinecone, און דערצו דער requirements טעקע.

אין דער .env טעקע לייגט מען אריין:

- דעם API key
- דעם index נאמען. אין דער דעמא ווערט געשאפן אן index מיטן נאמען "semantic-search-demo". דעם index שאפט מען אויף Pinecone: ביים "Getting Started" בעט מען צו שאפן אן index, און אונטן איז דא א קנעפל "Create Index".

- דעם cloud provider (AWS)
- די ראיאן (us-east-1)

דער .env טעקע היט אפ אלע environment variables וואס מען דארף.

requirements און א virtual environment (14:09:28)

דער requirements טעקע רעכנט אויס די ביבליאטעקן מיט זייערע ווערסיעס: uvicorn, FastAPI, python-dotenv און pinecone-client.

דערנאך שאפט מען א virtual environment דורכן terminal אין VS Code. דעם באפעל געפינט מען אין דער אפיציעלער דאקומענטאציע פון Python:

m venv venv- python

דאס שאפט א פאלדער "venv", וואס איז דער virtual environment. עס ווערט אונטערגעשטראכן אז עס איז א גוטע פירונג אנצוהייבן אזוי. נאכדעם וואס דער פאלדער איז געשאפן, דארף מען אקטיווירן דעם virtual environment.

ערקלערונג פון די ביבליאטעקן אין requirements.txt:

- pinecone-client, ווערסיע 2.2.2: פארבינדט Python מיט Pinecone.
- python-dotenv: לייענט די סודות-ווערטן, ווי API keys, פונעם .env טעקע.
- FastAPI: צו שאפן APIs.
- uvicorn: דער סערווער אויף וועלכן די אפליקאציע לויפט.

פארוואס היט מען די ווערטן אין .env? צוליב זיכערהייט. מען זאל זיי נישט שרייבן גלייך אינעם קאוד (hardcode), ווייל דאס קען ברענגען א זיכערהייט-פראבלעם.

דער קלאס PineconeOperations (14:11:15)

מען שאפט א קלאס מיטן נאמען PineconeOperations. ער איז וויכטיג פאר די פעולות: שאפן אן index, זיך פארבינדן מיט אן index, אריינלייגן וועקטארן, קוקן די סטאטיסטיק פונעם index, און זוכן ענליכע וועקטארן.

דער קאוד ארבעט אזוי, שריט ביי שריט:

- מען אימפארטירט load_dotenv פון dotenv, און Pinecone און ServerlessSpec פון pinecone.
- דער קלאס לאדנט אלע environment variables פונעם .env טעקע.
- מען נעמט ארויס דעם API key, דעם index נאמען, דעם cloud און די ראיאן.
- אויב דער API key איז נישטא, וואַרפט מען א טעות מיט דער מעלדונג "Please add a valid API key".
- צו פארבינדן זיך מיט Pinecone שאפט מען א Pinecone client מיטן נאמען pc, מיטן API key.
- דער index אביעקט ווערט ערשט געשאפן נאכן פארבינדן זיך, דערפאר שטעלט מען אנהייבס self.index = None.

די מעטאד create_index: dimension און cosine (14:12:40)

מען שאפט א מעטאד create_index מיט א פאראמעטער dimension = 8, און metric = "cosine". עס ווערט ערקלערט אז dimension מיינט אז יעדער וועקטאר מוז אנטהאלטן אכט נומערן, און Pinecone וועט פארגלייכן די ווערטן לויט cosine. דאס איז ענליך צו וועקטארן וואס מען לערנט אין דער 12טער קלאס: ווען מען האט צוויי וועקטארן, באניצט מען זיך מיט דער cosine פארמל צו פארגלייכן ווי ענליך זיי זענען.

דער סדר פון דער מעטאד:

- אויב self.index_name איז נישט צווישן די עקזיסטירנדע indexes, שאפט מען אן index מיטן נאמען, dimension, metric, און א serverless קאנפיגוראציע מיטן cloud און דער ראיאן.
- מען ווארט ביז דער index ווערט גרייט: א לופ וואס פרעגט describe_index, און ביז דער status איז נישט "ready", מאכט מען time.sleep(1).
- נאכן שאפן גיט מען צוריק א מעלדונג מיטן index נאמען.
- אויב דער index עקזיסטירט שוין, גיט מען צוריק א מעלדונג "index already exists" מיטן נאמען פונעם index.

עס ווערט אונטערגעשטראכן אז דער קלאס איז זייער וויכטיג, ווייל אלע פעולות וואס מען האט דורכגערעדט פאסירן אין אים.

דערנאך הייבט מען אן פארבינדן די אפליקאציע מיטן Pinecone index דורך א מעטאד וואס גיט צוריק א dictionary.

Pinecone מיט FastAPI, און אריינפיר אין RAG (14:14:15)

אפעראציעס אויף א Pinecone אינדעקס (14:14:15)

דער קורס האלט אינמיטן א פראיעקט וואס פארבינדט זיך מיט Pinecone, א וועקטאר-דאטאבאזע. מען האט שוין געזען ווי מען שאפט דעם אינדעקס. ווען די פארבינדונג מיט Pinecone איז געלונגען, גיט די פונקציע צוריק א מעלדונג מיטן נאמען פונעם אינדעקס.

דער נעקסטער שריט איז אריינלייגן אדער דערהיינטיגן וועקטארן אין Pinecone. דערפאר באניצט מען זיך מיט דער מעטאד upsert. ס'ווערט געוויזן א ביישפיל-אינפוט:

- ID: course1
- values: די צאלן פונעם וועקטאר
- metadata: דער title איז "Python for data science", און דער category איז "data science"

אינעם ענטפער זעט מען אז מען לייגט צו די וועקטארן צוזאמען מיט א namespace. דערנאך ציט מען ארויס סטאטיסטיק וועגן דעם Pinecone אינדעקס. ווען מען גרייכט צום אינדעקס, קומט צוריק א מעלדונג אז דער אינדעקס איז אריסגעצויגן געווארן.

אזוי אויך ביים זוכן: מען שאפט א ביישפיל query-וועקטאר, און דער ענטפער ווערט געהאלטן אין א ווערייאבל response. ערשט דארף מען זיך פארבינדן מיטן אינדעקס, און דאס איז זייער וויכטיג. דאן קומען צוריק די top

k רעזולטאטן, צוזאמען מיט די values, די metadata און דעם namespace. צום סוף גיט מען צוריק א מעלדונג אז דאס זוכן איז דורכגעגאנגען מיט הצלחה.

דער הויפט-פייל: API-וועגן מיט FastAPI און Pydantic (14:15:19)

דערנאך גייט מען איבער צום הויפט-פייל, main.py, וואו מען בויט די API-וועגן (routes). מען באניצט זיך מיט FastAPI, און אויך מיט Pydantic-מאדעלן, וואס זענען זייער נוצלעך אין אזא פאל.

- ערשט שאפט מען דעם app מיטן FastAPI קאנסטרוקטאר, און מען גיט אים א title, א description און א נומער פון וועדסיע.
- דערנאך שאפט מען אן אביעקט פונעם קלאס PineconeOps, וואס מען האט פריער געבויט פאר אלע אפעראציעס.
- מען דעפינירט א קלאס VectorData, וואס ירשנט BaseModel פון Pydantic. ער האט די פעלדער id, values און metadata.
- דער upsert-בקשה האט א payload וואס האלט די VectorData אין א ליסטע.
- נאך א קלאס, SearchRequest, ירשנט אויך פון Pydantic. ער האלט דעם וועקטאר און דעם צאל top k פון די ענליכסטע רעזולטאטן וואס מען וויל צוריקבאקומען.

דערנאך קומען די דעקאראטארן:

- א GET-בקשה פאר א health check, ווי אן async פונקציע. זי גיט צוריק א מעלדונג אז די אפליקאציע לויפט.
- א POST-בקשה אויפן וועג api/v1/index צו שאפן אן אינדעקס דורך PineconeOps.
- א בקשה צו פארבינדן זיך מיטן אינדעקס, דורך PineconeOps connect.
- א GET-בקשה וואס ברענגט די סטאטיסטיק פונעם אינדעקס.
- אן async מעטאד פאר upsert פון וועקטארן. דארט לייגט מען צו id, values און metadata אין א ליסטע פון וועקטארן, און מען רופט די PineconeOps מעטאד מיט דער דאטא.
- א POST-בקשה אויפן וועג פון search vector, אויך async. זי זוכט די query און גיט צוריק די top k רעזולטאטן.

דער קורס באטאנט אז דאס איז א גאנץ פשוטע אפעראציע: מען פארבינדט זיך מיט Pinecone און מען באקומט צוריק רעזולטאטן וואס זענען ענליך לויטן באדייט (semantically similar).

וואס איז אסינכראניש? (14:16:46)

אויב מען איז נישט באקאנט מיט אסינכראנישע פראצעסן, רעקאמענדירט דער קורס צו לייענען דערוועגן אין דער אפיציעלער דאקומענטאציע. דער ביישפיל: מען שיקט א בקשה צו א וועטער-API צו באקומען דעם וועטער אין San Francisco. דער רעזולטאט קען נעמען א ביסל צייט, און די מעטאד וואלט ביז דער ענטפער קומט אן. בקיצור, אן אסינכראנישער פראצעס איז איינער וואו ס'איז פארמישט א צייט פון וואלטן, און מען לייגט דעם ענטפער אריין אין א וועריאבל. מען באניצט זיך מיט async און await, וואס מען טרעפט מערסטנס אין JavaScript. וועט ס'האט געלערנט async און await אין JavaScript, וועט זען אז דאס איז זייער ענליך.

אינסטאלירן, לויפן און טעסטן (14:18:47)

- צו אינסטאלירן די ביבליאטעקן שרייבט מען: pip install -r requirements.txt. דאס אינסטאלירט אלע אפהענגיקייטן. די פעלערן וואס האבן זיך פריער געוויזן ווייל ס'האבן געפעלט אפהענגיקייטן, זענען דערמיט אויסגעהיילט געווארן.
- מען לויפט דעם פראגראם מיט Python 3. דער בעפעל איז uvicorn main:app --reload, און ער שטעלט אן דעם סערווער.
- דער סערווער לויפט אויף 127.0.0.1, דאס הייסט localhost. מען קלייבט Chrome אלס בלעטערער.
- צו טעסטן דעם API באניצט מען זיך מיט Reqbin, אן אנליין געצייג צו טעסטן APIs . מען לייגט אריין metadata, title, category און courses, מען דריקט Send, און די דאטא ווערט צוגעלייגט אין Pinecone.

א נייער אפטייל: אריינפיר אין RAG (14:20:21)

דא הייבט זיך אן א נייער טייל פונעם קורס. דער פלאן: אן אריינפיר אין RAG; די באשטאנדטיילן פון RAG; לאדן דאטא און צעטיילן טעקסט; אפהיטן די שטיקלעך (chunks) אין וועקטאר-סטארס מיט embeddings; ארויסציען די דאטא; בויען א מולטי-דאקומענט אגענט; און א סיום.

ס'ווערט געגעבן א ביישפיל: מען שמועסט מיט אן AI און מען פרעגט "ווער האט געוואונען דעם FIFA World Cup אין 2022?" דער AI קען נישט סתם טרעפן, ער דארף אמת'דיגע אינפארמאציע. אבער רוב שפראך-מאדעלן, ווי ChatGPT, זענען טריינירט נאר ביז א געוויסער צייט. וואס ס'איז געשען נאך זייער טריינירן, ווייסן זיי נישט. דא קומט RAG, דאס הייסט Retrieval Augmented Generation, און לייזט דעם פראבלעם.

RAG איז ווי מען וואלט געגעבן דעם AI א שטארקן חבר וואס איז א זוך-מאשין. אנשטאט זיך צו פארלאזן נאר אויף וואס ער געדענקט פונעם טריינירן, ציט ער אויך ארויס די נייסטע און מערסט שייכות'דיגע אינפארמאציע פון א פארבונדענעם מקור, ווי א דאקומענט, א וויסן-באזע אדער אפילו די וועב. דערנאך באניצט ער זיך דערמיט צו שאפן א בעסערן ענטפער. די פארמל: RAG = (Retrievalגעפינען אינפארמאציע) + Generation)שרייבן ענטפערס). ס'איז ווי א תלמיד וואס שרייבט א פראבע מיט אן אפענעם ספר: ער זוכט ערשט נאך, און דערנאך ערקלערט ער אין זיינע אייגענע ווערטער.

פארוואס איז דאס וויכטיג? טראדיציאנעלע שפראך-מאדעלן האבן א knowledge cutoff, דאס הייסט זיי ווייסן נישט פון גארנישט וואס איז געשען נאך א געוויסער דאטע. RAG:

- פארבעסערט די ענויקייט, ווייל ער ציט ארויס אמת'דיגע אינפארמאציע ווען מ'דארף.
- גיט דערהיינטיגטע ענטפערס, אפילו אויב דער AI איז טריינירט געווארן מיט יארן צוריק.
- פארמינערט hallucination, ווייל ער באניצט זיך מיט פאקטן אנשטאט צו טרעפן.
- מאכט AI מער פארלעסלעך אין קריטישע ארבעטן, ווי געזונט-אפהיטונג און enterprise search.

ווי מען באניצט RAG, און ווי ער ארבעט (14:22:57)

RAG איז זייער נוצלעך ווען געוויקייט און דערהיינטיגט וויסן זענען וויכטיג. די ביישפילן:

- שטיצע-באטס פאר קונים: זיי ציען ענטפערס פון הילף-דאקומענטן, FAQs אדער אינערליכע וויסן-באזעס.

- enterprise search : ארבעטער פרעגן און באקומען ענטפערס פון די פירמע-דאקומענטן, מאנואלן אדער HR-פאליסיס.
- געזונט-אפהיטונג: דאקטוירים פרעגן וועגן באהאנדלונג-אנווייזונגען, און RAG ציט אינפארמאציע פון נייע מעדיצינישע פארשונגס-ארבעטן.
- געזעץ: אדוואקאט-פירמעס געפינען און פאסן צווזאמען case laws פון א ריזיגן ארכיוו פון דאקומענטן.
- בילדונג: סטודענטן פרעגן, און דער AI געפינט דעם בעסטן ענטפער פון לערן-ביכער אדער נאטיצן.

ווי RAG ארבעט, אין פשוטע שריט:

- שריט 1: דער באניצער פרעגט, למשל "וואס זענען די נעבן-ווירקונגען פון ?aspirin"
- שריט 2, Retrieve: דער AI זוכט ערשט אין א פארבונדענער דאטאבאזע, ווי מעדיצינישע דאקומענטן, ארטיקלען אדער PDFs. דאס טוט מען מיט א retriever, ווי BM25 אדער dense vector search.
- שריט 3, Augment: מען קלייבט אויס די מערסט שייכות'דיגע דאקומענטן אדער שטיקלעך, און מען פארבינדט זיי מיט דער שאלה.
- שריט 4, Generate: דער שפראך-מאדעל, ווי GPT אדער BERT, נעמט דעם קאנטעקסט און שאפט דעם ענטפער לויט דער אינפארמאציע וואס ער האט ערשט ארויסגעצויגן.

די משל: א ביבליאטעקער ענטפערט נישט פון זכרון. ער גיפינט ערשט דאס ריכטיגע בוך (דער retriever), ער לייענט דעם ריכטיגן בלאט (די דאקומענטאציע), און ער ערקלערט דעם ענטפער אין פשוטע שפראך (די generation).

די הויפט-באשטאנדטיילן פון א RAG pipeline (14:25:01)

דער RAG pipeline איז נישט קיין איינציגער מאדעל. ער איז א קלוגע סיסטעם פון עטלעכע טיילן, און יעדער טייל שפילט אן אייגענע ראלע. ס'זענען דא זעקס הויפט-טיילן: query encoder, retriever, knowledge base (document store), reranker, generator (דער שפראך-מאדעל), און feedback loop.

- Query encoder: ער נעמט די שאלה פונעם באניצער, למשל "וואס זענען די מעלות פון גרינע טיי?", און ער פארוואנדלט זי אין א מאטעמאטישע פארעם וואס הייסט א וועקטאר. דערמיט קענען מאשינען זוכן לויט דער שאלה. אן encoding קען די סיסטעם נישט פארגלייכן די שאלה מיט דער אינפארמאציע אין דער וויסן-באזע. דער encoder זארגט אז די שאלה איז אין אזא פארמאט וואס דער retriever פארשטייט און קען צופאסן צום שייכות'דיגן אינהאלט.
- Retriever: ער באניצט זיך מיט דער אנקאדירטער שאלה צו זוכן אין דער וויסן-באזע, און ער ברענגט די בעסטע פאסנדע דאקומענטן אדער פאסאזשן. דא וועלט RAG שטארק: אנשטאט זיך צו פארלאזן אויפן זכרון פונעם מאדעל, ברענגט ער פרישע דאטא פון פארלעסלעכע מקורות. דאס העכערט די געניויקייט און פארמינערט hallucinations.
- Knowledge base (document store): א ביבליאטעק וואס האלט אלעס וואס מען קען דורכזוכן, ווי PDFs, שטיצע-דאקומענטן, מאנואלן און פארשונגס-ארבעטן. "דא וואוינט דער אמת." א גוט-אויסגעשטעלטע וויסן-באזע, וואס וועלט רעגולער דערהיינטיגט, זארגט אז דער AI-געהילף קען ארויסציען פרישע און פארלעסלעכע אינפארמאציע.

- Reranker: נאכדעם וואס דער retriever ברענגט די top N דאקומענטן, אפשאצט זיי דער reranker נאכאמאל און ער ארדנט זיי איבער לויט ווי שייך זיי זענען צו דער שאלה. אמאל ברענגט דער retriever גאנץ גוטע רעזולטאטן, אבער נישט אין דער בעסטער ארדנונג. דער reranker שפילט די ראלע פון א קוואליטעט-קאנטראלירער.

דאקומענט-לאדער און טעקסט-צעטיילן אין LangChain (14:28:09)

דער גענעראטאר און דער פידבעק-שלייף אין RAG (14:28:09)

דער קורס ענדיגט די באשרייבונג פון די טיילן פון א RAG סיסטעם. פריער איז ערקלערט געווארן דער reranker, וואס שטופט די נוצליכסטע דאקומענטן ארויף אויבן, נאך איידער דער גענעראטאר זעט זיי.

דער פינפטער טייל איז דער גענעראטאר, דאס הייסט א שפראך-מאדעל. מען רופט עס "די שטימע פון אייער AI". א גענעראטאר ווי GPT נעמט די שאלה צוזאמען מיט די בעסטע דאקומענטן, און שאפט דערפון אן ענטפער וואס קלינגט ווי א מענטש און גיט אינפארמאציע. דער קורס זאגט אז דא "געשעט דער כישוף": דער מאדעל טרעפט נישט מער לויט וואס ער האט זיך אויסגעלערנט ביים טריינירן. ער ניצט די פאקטן וואס מען האט ארויסגעזוכט, און דערמיט שאפט ער ענטפערס וואס זענען גענוי און באזירט אויף אמת'ע מקורות.

דער זעקסטער טייל איז דער פידבעק-שלייף. ער זאמלט די רעאקציעס פון די באניצער, למשל א דוים ארויף, א דוים אראפ אדער קליקס, און גיט דאס צוריק אריין אין סיסטעם. אזוי לערנט זיך דער RAG סיסטעם מיט דער צייט. וועז באניצער האבן כסדר ליב געוויסע ענטפערס אדער מקורות, פאסט זיך דער סיסטעם צו און גיט זיי פארצוג דאס קומענדיגע מאל. דער שלייף העלפט אויך צו טריינירן נאכאמאל אדער צו fine-tune 'ן דעם retriever און דעם reranker.

וואס זענען דאקומענט-לאדער (14:29:10)

א נייער קאפיטל הייבט זיך אן: לאדן און צעטיילן דאקומענטן. וועד עס וויל בויען אן אפליקאציע וואו מען קען "שמועסן מיט די אייגענע דאטא", מוז קודם אריינלאדן די דאטא אין א פארעם וואס מען קען מיט איר ארבעטן. דערפאר זענען דא די document loaders פון LangChain. עס זענען דא איבער 80 סארטן לאדער. אין דער לעקציע ווערן געוויזן נאר עטליכע, כדי מען זאל זיך באקענען מיטן געדאנק בכלל.

א לאדער באהאנדלט די פרטים ווי אזוי מען קומט צו דער דאטא און ווי אזוי מען פארוואנדלט זי פון פארשידענע פארמאטן און מקורות אין איין סטאנדארט פארמאט. די דאטא קען קומען פון וועבזייטלעך, דאטא-באזעס, YouTube און נאך. די דאקומענטן קענען זיין PDF, HTML, JSON, Word, PowerPoint און אזוי ווייטער. דער ציל פון די לאדער איז צו נעמען די אלע מקורות און זיי אריינלאדן אין א סטאנדארט document object. אזא אביעקט האט צוויי טיילן: דעם אינהאלט, און די metadata וואס געהערט צו אים.

מען קען נישט דורכגיין אלע לאדער, אבער דער קורס טיילט די דאטא גראב איין אין פיר גרופעס:

- פובליק אומסטרוקטורירט: דאטא וואס איז פריי צוגענגליך פאר יעדן, אבער האט נישט קיין פעסטן פארמאט. למשל פאסטן אויף סאשעל מידיע, בלאגס, ארטיקלען און YouTube ווידעאס.
- פובליק סטרוקטורירט: פריי צוגענגליכע דאטא וואס איז אויסגעשטעלט אין א קלארן פארמאט. למשל דאטא פון דער רעגירונג'ס צענזוס, בערזע-דאטא, וועטער-באריכטן.

• פריוואט אומסטרוקטורירט (proprietary): פריוואטע דאטא וואס געהערט צו פירמעס אבער איז נישט אויסגעשטעלט. למשל אימעילס פון קונדן-סערוויס, אינערליכע באריכטן פון דער פירמע, רעקארדירטע טעלעפאן-שמועסן.

• פריוואט סטרוקטורירט: פריוואטע, גוט-אויסגעשטעלטע דאטא פון פירמעס. למשל די טראנזאקציע-רעקארדס פון א באנק'ס קונדן, די היסטאריע וואס א באניצער האט געקוקט אויף Netflix, די פארקויף-דאטא-באזע פון Amazon.

OpenAI פרייזן און לאדן PDF דאקומענטן (14:31:44)

פאר די פראקטישע טייל ווייזט דער קורס וואו מען זעט די פרייזן פון דעם OpenAI API: אויפן זייטל פון API pricing אויף openai.com. וועט מען סקראלט אראפ קען מען אויספארשן די פרטים, און מען זעט אלע לעצטע מאדעלן מיט זייערע פרייזן פאר input, cached input און output. אין דעם קורס ניצט מען GPT-3.5 Turbo. דער קורס דערמאנט עטליכע ציפערן פאר אים: $13 פאר input און $18 פאר טריינירן, און לויטן פרייז-זייטל $0.5 פאר input און $1.50 פאר output. די ציפערן זענען נישט אינגאנצן קלאר אינעם רעקארדינג. מען קען אויסקלייבן וועלכן API עס פאסט אמבעסטן פארן אייגענעם פראיעקט.

צום אנהייב אימפארטירט מען די environment variables וואס מען דארף, ווי דעם OpenAI API key.

דער ערשטער סארט דאקומענט איז PDF:

• מען אימפארטירט פון LangChain דעם PyPDFLoader.

• אין דער workspace האט מען אריינגעלייגט א רייַ PDFs אין א documents טעקע, און מען קלייבט אויס איינעם.

• מען לאדט דעם דאקומענט דורכן רופן די מעטאד load.

• מען קוקט אויף דער לענג פון דער רשימה בלעטער: עס זענען 22 בלעטער, און יעדער בלאט איז א באזונדערער דאקומענט.

• דער אינהאלט פונעם ערשטן בלאט איז דער אנהייב פון א לעקציע: "Good …the instructor is Andrew morning. Welcome to CS229, the machine learning class."

• די metadata האט צוויי פעלדער: source, דאס הייסט דער נאמען פון דער PDF טעקע פון וואנען מען האט גענלאדן, און page, די נומער פונעם בלאט אין דער PDF.

לאדן אינהאלט פון YouTube (14:33:51)

דער קומענדיגער סארט אינהאלט איז YouTube. דארט איז פאראן א סך אינטערעסאנטער אינהאלט, און א סך מענטשן ניצן דאקומענט-לאדער כדי צו פרעגן שאלות וועגן לעקציעס און ענליכס. די הויפט-טיילן וואס מען אימפארטירט:

• YoutubeAudioLoader: לאדט די אודיא-טעקע פון א YouTube ווידעא.

• OpenAIWhisperParser: ניצט OpenAI'ס Whisper מאדעל. דאס איז א speech-to-text מאדעל, וואס פארוואנדלט דאס אודיא אין טעקסט, אזוי אז מען קען מיט אים ארבעטן.

די שריט: מען שטעלט אריין דעם URL פונעם ווידעא. מען ג'ט אן א טעקע וואו מען זאל אפהיטן די אודיא-טעקעס. דערנאך שאפט מען א GenericLoader וואס פאראייניגט דעם YoutubeAudioLoader מיטן

OpenAIWhisperParser. צום סוף רופט מען ()loader.load. דאס נעמט עטליכע מינוט. נאכדעם קען מען קוקן אויפן page_content פונעם ערשטן דאקומענט, און מען זעט דעם ערשטן טייל פונעם טראנסקריפט פונעם ווידעא.

לאדן וועב-זייטלעך (14:34:52)

אויפן אינטערנעט איז פאראן א סך לערן-אינהאלט, און עס וואלט געווען שיין צו קענען "שמועסן" מיט אים. דערפאר אימפארטירט מען דעם WebBaseLoader פון LangChain. מען קען אויסקלייבן וועלכן URL מען וויל. אין דער לעקציע האט מען גענומען א markdown טעקע פון א GitHub זייטל. מען רופט דעם לאדער-קלאס און קוקט אויפן אינהאלט. מען זעט דארט א סך ליידיגע פלאץ, דערנאך אביסל אינהאלט, ווידער ליידיג, און ווידער אינהאלט. דער קורס באמערקט אז דאס איז גאר אינטערעסאנט.

לאדן דאטא פון Notion (14:35:54)

צום סוף ווייזט דער קורס ווי אזוי מען לאדט דאטא פון Notion . מען דארף נאכגיין געווייסע שריט אויפן Notion זייטל:

- מען קאפירט (duplicate) דאס בלאט אין דער אייגענער Notion שטח.
- מען עקספארטירט עס אלס markdown אדער CSV.
- מען פאקט עס אויס (unzip) און מען היט עס אפ אלס טעקע וואס אנטהאלט די markdown טעקע פונעם Notion בלאט.

דערנאך ניצט מען דעם NotionDirectoryLoader כדי צו לאדן די דאטא און באקומען דאקומענטן. דער ערשטער בלאט איז אין ביישפיל איז א "לעבעדיגער דאקומענט" מיט אלעס וואס מען האט זיך אויסגעלערנט ביים ארבעטן מיט מענטשן וועו מען פירט א startup, און מען לערנט זיך ווייטער. עס איז א markdown העפט: אן employee handbook פון א פירמע (Blendle). דער קורס באמערקט אז א סך מענטשן ארבעטן מיט Notion און האבן א Notion דאטא-באזע, און דאס איז א גוטע געלעגנהייט צו עקספארטירן די דאטא, זי אריינברענגען אהער און אנהייבן מיט איר ארבעטן. אין דער metadata זעט מען דעם נאמען פונעם employee handbook.

פארוואס צעטיילן דאקומענטן (14:37:25)

נאכדעם וואס די דאטא איז געלאדן, גייט מען איבער צו צעטיילן (splitting). דאס קלינגט גרינג, אבער עס זענען דא א סך פיינע פרטים. דאס צעטיילן געשעט נאכן לאדן די דאטא אין document פארמאט, אבער איידער זי גייט אריין אינעם vector store. אויפן ערשטן בליק קען מען שניידן די שטיקלעך (chunks) פשוט לויט א צאל אותיות.

דער קורס ברענגט א ביישפיל מיט א זאץ: "the Honda Accord boasts an impressive 192HP engine paired with a seamless 10-speed automatic transmission, delivering a smooth yet powerful ride that will leave a lasting impression". שניידט מען עס אין צוויי, איז שטיקל איינס "the Honda Accord boasts", און שטיקל צוויי איז דער רעשט: "an impressive 192HP engine… lasting impression". וועו מען ווייל ענטפערן אויף דער שאלה "וואס זענען די ספעציפיקאציעס פון דער ?Honda", קען קיין איינס פון די שטיקלעך אליין נישט ענטפערן. דערפאר איז זייער וויכטיג ווי אזוי מען צעטיילט, כדי צו באקומען די רעלעוואנטע שטיקלעך אינפארמאציע.

די באזיס פון אלע text splitters אין LangChain איז צו צעטיילן אין שטיקלעך פון א געוויסע גרייס (chunk size), מיט א געוויסער איבערלאפ צווישן זיי (chunk overlap).

סארטן splitters אין LangChain (14:38:56)

- CharacterTextSplitter: צעטיילט לויט א געוויסן צייכן, למשל \n, מיט א chunk size און overlap.
- MarkdownHeaderTextSplitter: צעטיילט markdown דאקומענטן און היט אפ די כעדערס און די סטרוקטור.
- TokenTextSplitter: צעטיילט לויט דער צאל טאָקענס, כדי עס זאל פאסן פאר LLMs ווי GPT.
- SentenceTransformersTokenTextSplitter: ניצט sentence transformers כדי צו צעטיילן אויף א קלוגן אופן, לויטן באדייט.
- RecursiveCharacterTextSplitter: צעטיילט היראַרכיש. קודם לויט גרויסע איינהייטן, און דערנאך לויט קלענערע, ווי זאצן און וועדטער.
- Language: צעטיילן וואס רעכנט זיך מיט דער שפראך, לויט שפראכלעכע כללים.
- NLTKTextSplitter: ניצט NLTK כדי צו צעטיילן אין זינפולע שטיקלעך באזירט אויף זאצן.
- SpacyTextSplitter: ניצט די NLP מאדעלן פון spaCy, און היט אפ די גראמאטישע סטרוקטור.

א דעמא פון צעטיילן (14:40:28)

מען אימפארטירט os, openai און sys. פון dotenv אימפארטירט מען load_dotenv, און מען שטעלט איין דעם OpenAI API key. פון langchain.text_splitter אימפארטירט מען RecursiveCharacterTextSplitter און CharacterTextSplitter.

מען שטעלט chunk_size = 26 און chunk_overlap = 4. דערנאך שאפט מען צוויי splitters:

r_splitter = RecursiveCharacterTextSplitter(chunk_size=chunk_size, chunk_overlap=chunk_overlap)

c_splitter = CharacterTextSplitter(chunk_size=chunk_size, chunk_overlap=chunk_overlap)

מען גיט א ערשטן טעקסט צום r_splitter, און מען זעט אז עס איז נישטא קיין חילוק, ער ווערט נישט צעטיילט. מען נעמט א צווייטן, לענגערן ביישפיל, און דא זעט מען אז דער טעקסט ווערט יא צעטיילט. דער קורס באמערקט אבער אז כאטש דער overlap איז 4, זעט עס אויס ווי 3.

א דריטער ביישפיל: מיטן r_splitter קומען ארויס שטיקלעך ווי "a b c d…", "l m n o p q r s t u v w x…" און "w x y z", מיט איבערלאפ צווישן זיי. דער זעלבער טעקסט מיטן c_splitter ווערט ווידער צוזאמענגעשטעלט אין איין גאנצן טעקסט, ווייל דער CharacterTextSplitter צעטיילט לויט א געוויסן צייכן. דערנאך לייגט מען צו צום c_splitter א separator פון א ליידיגן פלאץ (space), אינעם קאוד אנגעצייכנט צווישן אנפירונגס-צייכנס. דאן צעטיילט ער דעם טעקסט ענליך ווי פריער, אבער לויט דעם separator, כדי מען זאל בעסער פארשטיין. דער קורס רעקאמענדירט אז דאס איז א גוטע צייט אפצושטעלן און אליין פרובירן אייגענע ביישפילן.

שפּאַלטן טעקסט, embeddings און וועקטאר-סטארס (14:43:07)

וויטער מיט CharacterTextSplitter און RecursiveCharacterTextSplitter (14:43:07)

אין דער לעקציע וועט וויטער געוויזן א ביישפיל, דאס מאל מיט א גאנצן פאראגראף. מען קוקט קודם אויף דער לענג פונעם טעקסט, און עס קומט אויס 496 וועט וויטער. דערנאך נעמט מען נאכאמאל די צוויי שפּאַלטערס: דעם "C splitter" (CharacterTextSplitter) און דעם "R splitter" (RecursiveCharacterTextSplitter), און מען איניציאליזירט זיי פון ניי.

- ביים C splitter איז דער chunk size 450, קיין chunk overlap איז נישטא, און דער separator איז א פּאָר גענשעפּיסלעך (inverted commas).
- ביים R splitter איז דער chunk size אויך 450, אויך אן overlap, און די separators זענען: צוויי נייע שורות, איין נייע שורה, גענשעפּיסלעך, און אזוי וויטער.

מען לויפט ערשט דעם C splitter אויפן טעקסט וואס מען האט צוגעגרייט, און מען זעט אז ער האט געשפּאַלטן לויט די גענשעפּיסלעך. דערנאך לויפט מען דעם R splitter אויף דעמזעלבן טעקסט, און מען זעט ווי ער האט אויך געשפּאַלטן ביי די גענשעפּיסלעך.

דערנאך וועט דער chunk size פארקלענערט, און מען לייגט צו א פונקט (period) צו די separators. אזוי זעט מען דעם אונטערשייד וואס די נייע separators מאכן אינעם רעזולטאט.

שפּאַלטן א PDF און א Notion דאטא-באזע (14:44:10)

איצט וועט גלאדן א PDF פון "machine learning lecture one", און מען רופט דעם loader קלאס כדי צו באקומען די pages. מען אימפּאָרטירט CharacterTextSplitter פון LangChain text splitters, מיט די פאלגנדע איינשטעלונגען:

- separator: "\n" (א נייע שורה)
- chunk size: 1000
- chunk overlap: 150

מען לויפט דעם קאוד און רופט text_splitter.split_documents(pages). דער רעזולטאט: די צאל דאקומענטן נאכן שפּאַלטן איז 77, בשעת די צאל pages איז 22.

דערנאך אימפּאָרטירט מען NotionDirectoryLoader, מען גיט אים א Notion דאטא-באזע און מען לאדט זי. מען לויפט דעמזעלבן text splitter אויף דער Notion דאטא-באזע. די לענג פון notion_db קומט אויס 52, און נאכן שפּאַלטן זענען דא 353 דאקומענטן.

שפּאַלטן לויט טאָקענס (TokenTextSplitter) (14:45:41)

עס וועט ערקלערט אז מען קען, וועט מען וויל, שפּאַלטן בפירוש לויט דער צאל טאָקענס. מען אימפּאָרטירט TokenTextSplitter פון LangChain, מיט chunk size 1 און chunk overlap 0. מען נעמט א טעקסט text1, וואס האט טאקע נישט קיין זין, אבער פארן ביישפיל איז עס גוט גענוג, און מען לויפט אים דורך text_splitter.split. מען זעט ווי דער גאנצער טעקסט וועט צעשפּאַלטן, אויך מיט געזונדערטע גענשעפּיסלעך.

דערנאך לויפט מען דאס אויף די pages און מען קוקט אויפן ערשטן דאקומענט. ער באשטייט פון page_content און metadata. די metadata אנטהאלט דעם) sourceדי לעקציע) און די page, אזוי ווי עס איז שוין פריער באשריבן געווארן. צום סוף פונעם טאָקען-שפּאַלטן קוקט מען אויף דער metadata פון א בלאט, און מען זעט אז עס איז דער machine learning lecture one PDF מיט דער נומער פונעם בלאט.

שפּאַלטן לויט קאנטעקסט: MarkdownHeaderTextSplitter (14:46:45)

איצט קומט "קאנטעקסט-שפּאַלטן". מען אימפּאָרטירט NotionDirectoryLoader און MarkdownHeaderTextSplitter. מען האט א markdown דאקומענט, און מען באשטימט headers_to_split_on:

- Header 1 = "#"
- Header 2 = "##"
- Header 3 = "###"

מען שאפט דעם markdown_splitter און מען שרייבט:

markdown_splitter.split_text(markdown_document) = md_header_splits

מען קוקט אויף [0]md_header_splits, דער ערשטער דאקומענט. דער אינהאלט איז "Hi this is Jim … Hi this is Joe", און די metadata ווייזט דעם header title און Header 2 "Chapter 1". דער צווייטער דאקומענט אנטהאלט "Hi this is Lance", מיט Chapter 1 און Header 3 "Section".

דערנאך פראבירט מען דאס אויף דער נייער Notion דאטא-באזע. דאס מאל זענען אין headers_to_split_on דא בלויז Header 1 און Header 2. מען איניציאליזירט נאכאמאל MarkdownHeaderTextSplitter, מען לויפט markdown_splitter.split_text אויפן ערשטן טעקסט, און מען קוקט אויפן ערשטן רעזולטאט.

א נייער קאפיטל: chunks, embeddings און וועקטאר-סטארס (14:48:22)

דער קורס הייבט אן א נייעם קאפיטל: ווי מען האלט אפ די chunks, וואס זענען וועקטאר-סטארס, און וואס זענען embeddings. עס ווערט געגעבן א משל: שטעלט אייך פאר אז איר לייענט א בוך פון 200 בלעטער און איר ווילט געדענקען די וויכטיגסטע אידייען. אנשטאט אויסלערנען יעדעס וואָרט, צייכנט מען אונטער די הויפּט-זאצן און מען שרייבט זיי אויף קליינע קלעפּ-צעטלעך. די צעטלעך זענען די chunks.

אין דער וועלט פון AI און גרויסע שפּראך-מאדעלן (LLM), ספּעציעל אין RAG סיסטעמען (retrieval augmented generation), טוט מען כמעט דאסזעלבע. מען צעשפּאַלט גרויסע דאקומענטן אויף קליינע, באדייטפולע שטיקלעך טעקסט. אזוי קען דער מאדעל אריסנעמען נאר די טיילן וואס ער דארף, אנשטאט לייענען אלעס ווידער און ווידער.

פארוואס איז וויכטיג צו האלטן אין ?chunks מען קען דאך פרעגן: פארוואס נישט אפהאלטן דעם גאנצן דאקומענט? דאס איז א גערעכטע שאלה. דער ענטפער איז אזוי:

- א LLM קען אויף איין מאל באארבעטן נאר א באגרעניצטע מאס טעקסט. גיט מען אים א גאנץ בוך, קען ער ווערן איבערגעפילט, אדער ער קען איבערזען וויכטיגע טיילן.
- ווען די אינפארמאציע איז אפגעהאלטן אין קלענערע chunks, איז גרינגער פארן מאדעל צו געפינען געשווינד דעם פאסיגן אינהאלט.

- קלענערע chunks פארבעסערן די ענויקייט, ווייל דער מאדעל קאנצענטרירט זיך אויף ספעציפישע פרטים אנשטאט געבן אומקלארע ענטפערס.

וואס זענען embeddings און וועקטאר-סטארס (14:49:57)

דא קומט דער אינטערעסאנטער טייל: ווי געפינט מען שפּעטער דעם ריכטיגן ?chunk דערפאר זענען דא embeddings. אן embedding פארוואנדלט ווערטער אין נומערן וואס כאפּן אריין דעם באדייט. מען קען זיי פארגלייכן מיט GPS קאָאָרדינאטן פאר אידייען. למשל, די ווערטער "dog" און "puppy" וועלן זיין נאענט איינס צום אנדערן אינעם embedding-שטח, ווייל זייער באדייט איז ענלעך.

א ביישפיל: לאמיר נעמען דעם chunk "Python is a programming language used for data science". מען פארוואנדלט אים אין א רשימה נומערן, דאס הייסט אן embedding, למשל 0.12, 0.98, 0.34- און נאך. דער וועקטאר שטעלט פאר דעם באדייט פונעם זאץ, אבער נישט אין ווערטער.

אבער וואו האלט מען אפ אלע די באדייט-וועקטארן? אין א וועקטאר-סטאר. מען קען זיך עס פארשטעלן ווי א קלוגע ביבליאטעק: אנשטאט סדרן ספרים לויט די טיטלען, סדרט זי chunks לויט זייער באדייט. וועו מען שטעלט א שאלה, פארגלייכט זי דעם embedding פון דער שאלה מיט אלע אפּגעהאלטענע embeddings, און זי ציט ארויס די מערסט פאסיגע chunks.

א ביישפיל: מען פרעגט "What is Python used for?" די סיסטעם פארוואנדלט די שאלה אין אן embedding און זוכט אינעם וועקטאר-סטאר דעם נאענטסטן טרעף. דאס קען זיין דער chunk וואס מען האט פריער אפּגעהאלטן, און דער ענטפער איז: Python איז א פּראָגראַמירונג-שפּראך וואס מען ניצט פאר דאטא-וויסנשאפט.

ווי אלעס ארבעט צוזאמען אין א RAG סיסטעם (14:51:29)

וועו מען לייגט אלעס צוזאמען, זענען די שריט אזוי:

- מען צעשפּאַלט דעם דאקומענט אויף chunks.
- מען שאפט embeddings פאר די chunks.
- מען האלט אפ די embeddings אין א וועקטאר-סטאר.
- וועו א שאלה קומט אן, שאפט מען איר embedding.
- דער וועקטאר-סטאר געפינט די מערסט פאסיגע chunks.
- צום סוף ניצט דער LLM די chunks כדי צו שאפן א געוויען ענטפער.

אזוי ארבעט א RAG סיסטעם. לויט דעם קורס זענען אזעלכע סיסטעמען נישט נאר קלוג, נאר אויך עפעקטיוו: זיי פרובירן נישט אויסלערנען אלעס. אנשטאט דעם האלטן זיי אפ דאס וואס איז וויכטיג, אינעם ריכטיגן פארמאט, און נעמען עס ארויס וועו מען דארף.

דעמא: שפּאַלטן די CS229 לעקציעס און שאפן embeddings (14:52:00)

מען הייבט אן מיט שטעלן די environment variables. מען לאדט די זעלבע דאקומענטן ווי פריער און מען אימפּאָרטירט RecursiveCharacterTextSplitter, מיט chunk size 1500 און chunk overlap 150. עס ווערט צוגעלייגט או די PDFs זענען א סעריע CS229 לעקציעס, און או lecture one איז בכוונה אריינגעלייגט צוויי מאל, כדי צו סימולירן "שמוציגע" דאטא. נאכן לויפן דעם splitter זענען דא 209 chunks.

דערנאך נעמט מען די splits און מען שאפט embeddings מיט OpenAI. אבער פארן פראבירן אויף ריכטיגע דאטא, פראבירט מען קודם דריי קליינע זאצן:

- sentence1: "I like dogs"
- sentence2: "I like canines"
- sentence3: "The weather is ugly outside"

מען שאפט אן embedding פאר יעדן זאץ און מען אימפאָרטירט NumPy. מען ערווארט אז די ערשטע צוויי זאצן וועלן זיין זייער ענלעך. כדי צו פארגלייכן ניצט מען דעם dot product. עס ווערט ערקלערט אז דער dot product מולטיפליצירט די צוגעפאַסטע עלעמענטן פון צוויי וועקטארן און לייגט זיי צוזאמען, אזוי אז מען באקומט איין נומער. ער מעסט ווי ענלעך צוויי embeddings זענען: א העכערע ווערט מיינט מער ענלעכקייט, און א נידעריגערע ווערט מיינט ווייניגער.

דער רעזולטאט: דער dot product פון זאץ 1 און 2 איז היפש העכער ווי פון זאץ 1 און 3, וואס קומט אויס באדייטנד נידעריגער. דער dot product פון זאץ 2 און 3 איז כמעט די זעלבע נידעריגע ווערט ווי ביי 1 און 3. עס ווערט גאראנטירט אפצושטעלן דעם ווידעא און פראבירן אנדערע ביישפילן מיט זייערע dot products.

Chroma וועקטאר-סטאר און similarity search (14:55:04)

איצט שאפט מען embeddings פאר אלע chunks פון די PDFs און מען האלט זיי אפ אין א וועקטאר-סטאר. מען ניצט Chroma, וואס מען אימפאָרטירט פון LangChain vectorstores . עס ווערט ערקלערט אז LangChain האט אינטעגראציעס מיט א סך וועקטאר-סטארס. מען קלייבט דא Chroma ווייל זי איז לייכט און ארבעט אין זכרון (in-memory), און דערפאר איז זייער גרינג אנצוהייבן מיט איר. עס זענען דא אנדערע וועקטאר-סטארס וואס באטן אן כאָסטירטע לייזונגען. די קענען זיין נוצלעך ווען מען וויל אפהאלטן גרויסע מאסן דאטא, אדער אפהאלטן זיי אין קלאוד.

מען וויל דעם וועקטאר-סטאר אפהאלטן פאר די קומענדיגע לעקציעס. דערפאר שאפט מען א וועריאבל persist_directory מיט דעם וועג docs/chroma, און מען מאכט זיכער אז דארט איז נאך גארנישט פארהאן. דערנאך שאפט מען דעם וועקטאר-סטאר מיט Chroma.from_documents, וואס באקומט די splits פון פריער, די OpenAI embeddings, און דעם persist_directory . מען לויפט דעם קאוד און קוקט אויפן collection count: די צאל איז די זעלבע ווי די צאל splits פון פריער.

איצט הייבט מען אן ניצן דעם וועקטאר-סטאר. מען דארף אויסטראכטן א שאלה. וויבאלד דאס זענען לעקציעס פון א קלאס, קען מען פרעגן צי עס איז דא אן אימעיל אדרעס פאר הילף. די שאלה איז: "is there an email I can ask for help". מען ניצט דעם similarity_search מעטאד, מען גיט אים די שאלה, און k=3. דער k באשטימט וויפל דאקומענטן מען וויל צוריקבאקומען.

ריטריוועל אין RAG: פעלערן, MMR און מער (14:57:05)

די פראבלעמען פון סעמאנטישע זוכן (14:57:05)

מען לויפט דעם קאוד און מען קוקט נאך די לענג פון די דאקומענטן וואס זענען צוריקגעקומען. עס ווערט געוויזן אז די לענג איז דריי, פונקט ווי מען האט עס פארגעשריבן. דערנאך קוקט מען אריין אין דעם אינהאלט פונעם

ערשטן דאקומענט. עס רעדט פון אן אימעיל-אדרעס צו וועלכן מען קען שיקן פראגעס. אין טעקסט שטייט אז דער אימעיל גייט צו א חשבון וואס אלע) TAsדי געהילפן פונעם קלאס) און דער פראפעסאר לייענען.

נאך דעם ווערט באויזן ווי אזוי מען היט אפ (persist) די דאטא-באזע, כדי מען זאל זי קענען נוצן שפעטער.

דער קורס ערקלערט אז similarity search גיט בערך 80% פון די גוטע רעזולטאטן. עס זענען אבער פאראן געוויסע "עדזש-קעיסעס", ספעציעלע פעלער וואו עס פעלט אויס. דער ווייטערדיגער טייל באהאנדלט נאר דאס.

ערשטער פעלער: מען פרעגט "What did they say about MATLAB" און מען פירט דורך א similarity search מיט k=5. מען קוקט אויפן ערשטן דאקומענט און דערנאך אויפן צווייטן, און מען זעט אז מען באקומט דופליקאטן, די זעלבע שטיקלעך (chunks) צוויי מאל. די סיבה איז אז דער PDF פון דער ערשטער מאשין-לערנען לעקציע איז צוויי מאל אריין אין אינדעקס. דער סעמאנטישער זוך ברענגט אלע ענליכע דאקומענטן, אבער ער זארגט נישט פאר פארשידנקייט. דעריבער זענען docs[0] און docs[1] אינגאנצן גלייך. דאס איז דער ערשטער "פעלער-אופן".

צווייטער פעלער: מען פרעגט "What did they say about regression in the third lecture?" דאס קען מען קאנטראלירן, ווייל צו יעדן דאקומענט איז פאראן מעטאדאטא וואס זאגט פון וועלכער לעקציע ער שטאמט. מען ניצט ווידער similarity search אויף דער vector DB מיט k=5, מען גייט דורך די דאקומענטן אין א שלייף, און מען דרוקט אויס די מעטאדאטא. מען זעט אז עס קומט א געמיש: טייל רעזולטאטן פון דער דריטער לעקציע, און טייל פון דער צווייטער און ערשטער.

עס ווערט ערקלערט פארוואס דאס פעלט אויס: "דריטע לעקציע" איז א שטיק סטרוקטורירטע אינפארמאציע, אבער מען טוט א סעמאנטישן זוך לויט embeddings. דער embedding ווערט געשאפן פאר דעם גאנצן זאץ, און ער איז ווארשיינליך מער פאקוסירט אויף "regression". אז מען קוקט אויפן פינפטן דאקומענט, דער וואס קומט פון דער ערשטער לעקציע, זעט מען אז ער רעדט טאקע פון regression. דער זוך כאפט אלזא די טעמע, אבער נישט דעם פאקט אז ער דארף זוכן נאר אין דאקומענטן פון דער דריטער לעקציע, ווייל דאס איז סטרוקטורירטע אינפארמאציע.

דער קורס פארשלאגט דא אפצושטעלן, אליין פרובירן נאך פראגעס און טוישן דעם ווערט פון K. מיט דעם ענדיגט זיך די לעקציע וועגן די יסודות פון סעמאנטישן זוך און זיינע פעלער-אופנים. אין דער קומענדיגער לעקציע ווערט געזוכט א לייזונג פאר די פעלער.

וואס איז ריטריוועל, און צווייי נייע טעכניקן (15:00:10)

די נייע לעקציע גייט טיף אריין אין ריטריוועל און אין עטליכע מער-אוואנסירטע מעטאדן צו באוועלטיגן די עדזש-קעיסעס.

דעפיניציע: אין RAG מיינט ריטריוועל ארויסצונעמען די רעלעוואנטע דאקומענטן אדער דאטא פון א וויסן-באזע, למשל א vector database, א געוועלטיגע דאטא-באזע אדער לאקאלע פיילס, כדי צו ענטפערן אויף דעם באניצערס פראגע. ריטריוועל העלפט צו זוכן, צו פילטערן און צו געפינען אינפארמאציע אויף אן עפעקטיוו אופן.

וואו מען ניצט ריטריוועל:

- chatbots מיט זכרון, למשל צו געפינען פריערדיגע שמועסן.

- סיסטעמען וואס ענטפערן פראגעס, למשל צו געפינען ענטפערס פון FAQ.
- זוך-מאשינען באזירט אויף AI, פאר סעמאנטישן זוך.

עס ווערט געזאגט אז די טעכניקן וואס מען באהאנדלט דא זענען נייע אויפן מארק.

ערשטע טעכניק, Maximum Marginal Relevance (MMR): א קלוגער אופן אויסצוקלייבן אינפארמאציע וואס איז סיי רעלעוואנט און סיי פארשידנארטיג, פון א גרויסער צאל דאקומענטן. אנשטאט צו נעמען נאר די מערסט ענליכע רעזולטאטן, האלט MMR א באלאנס צווישן רעלעוואנץ און פארשידנקייט, אזוי אז די רעזולטאטן חזרן זיך נישט צופיל איבער. מען ניצט MMR אין RAG צו פארבעסערן די זוך-רעזולטאטן. א משל: מען זוכט "די בעסטע AI געצייג". אן MMR באקומט מען פינף כמעט גלייכע רעזולטאטן וועגן ChatGPT . מיט MMR באקומט מען רעזולטאטן וועגן ChatGPT, Gemini, Claude, Midjourney און Copilot.

צווייטע טעכניק, LLM-aided retrieval: א קלוגער אופן צו פארבעסערן זוך-רעזולטאטן דורכדעם וואס א גרויסער שפראך-מאדעל (LLM) פארפיינט אדער פירט דעם זוך-פראצעס. אנשטאט צו ברענגען דאקומענטן נאר לויט ענלעכקייט פון שליסל-וועדטער, העלפט דער LLM בעסער צו פארשטיין די פראגע, צו סדר'ן די רעזולטאטן לויט וויכטיגקייט און אויסצוקלייבן די רעלעוואנטע אינפארמאציע. א משל: אן דעם קען א זוך נאך "AI in marketing" ברענגען צופעליגע ארטיקלען וועגן AI. מיט LLM-aided retrieval פארשטייט דער LLM די כוונה און ברענגט ספעציפישע ארטיקלען וועגן AI גענעראל פאר מארקעטינג.

דעמא: אויפשטעלן און MMR אין אקציע (15:02:51)

די דעמא הייבט זיך אן ווי אלעמאל: מען לאדנט די environment variables . דערנאך אימפארטירט מען Chroma פון LangChain vectorstores און OpenAIEmbeddings, און מען ניצט דעם persist directory וואס מען האט פריער געשאפן, docs/chroma. מען שאפט embeddings מיט OpenAIEmbeddings, און מען שאפט די vector DB מיט Chroma, וואס באקומט דעם persist directory און די embedding function.

מען דרוקט אויס vectordb._collection.count(), און מען זעט 209 דאקומענטן. דאס זענען די וואס מען האט פריער אריינגעלאדנט.

א "צאצקע-ביישפיל" פאר MMR: עס איז פאראן א רשימה טעקסטן, אלע וועגן שוועמלעך (mushrooms). מען שאפט פון זיי א קליינע דאטא-באזע:

Chroma.from_texts(texts, embedding=embedding) = smalldb

די פראגע איז: "Tell me about all-white mushrooms with large fruiting bodies". מען רופט smalldb.max_marginal_relevance_search מיט דער פראגע, k=2 און fetch_k=3. דער קורס ווארנט צו געבן גענוי דעם ריכטיגן קאוד. עס ווערט ערקלערט אז k איז נאך אלץ 2, אבער fetch_k=3 מיינט אז מען ברענגט ערשט אלע דריי דאקומענטן, און דערנאך קלייבט מען אויס פון זיי.

דערנאך גייט מען צוריק צו דער MATLAB פראגע פון דער פריערדיגער לעקציע, "what did they say about MATLAB", מיט א similarity search אויף דער vector DB מיט k=3. מען פארגלייכט docs_ss[0].page_content מיט docs_ss[1].page_content, און מען זעט אז ביידע זענען גלייך. דערנאך ניצט מען MMR און מען פארגלייכט docs_mmr[0].page_content מיט docs_mmr[1].page_content. דאס מאל איז דער צווייטער אנדערש, און מען באקומט פארשידנקייט אין די רעזולטאטן.

Self-Query Retriever פאר מעטאדאטא (15:06:47)

מען לויפט ווידער די פראגע וועגן דער דריטער לעקציע מיט similarity search. דער רעזולטאט איז פונקט ווי פריער, און דער פעלער איז נאך אלץ דא.

די לייזונג איז SelfQueryRetriever. מען אימפארטירט OpenAI, SelfQueryRetriever און AttributeInfo. מען באשרייבט די מעטאדאטא-אטריבוטן מיט נאמען, באשרייבונג און טיפ. למשל "source" באקומט די באשרייבונג אז דער chunk שטאמט פון איינע פון די אויסגערעכנטע לעקציע-דאקומענטן, און "page" באקומט די באשרייבונג אז עס איז די זייט פון דער לעקציע.

דערנאך באשטימט מען דעם llm אלס אן OpenAI מאדעל, און מען שאפט דעם retriever מיט SelfQueryRetriever.from_llm, וואס באקומט דעם llm, די vector DB און די אנדערע פרטים. מען שטעלט verbose=True. מען שיקט אריין די פראגע, ברענגט די רעלעוואנטע דאקומענטן און דרוקט אויס די מעטאדאטא. א געוויסע ווארענונג וואס קומט ארויס קען מען איגנארירן. אין דער מעטאדאטא זעט מען פון וואנעט די דאקומענטן קומען.

קאנטעקסטועלע צוזאמענדריקונג (15:08:24)

אן אנדער שטייגער צו פארבעסערן די קוואליטעט פון די געפונענע דאקומענטן איז צוזאמענדריקונג (compression). די אינפארמאציע וואס איז מערסט רעלעוואנט פאר א פראגע קען זיין באגראבן אין א דאקומענט מיט א סך אומרעלעוואנטן טעקסט. אז מען שיקט דעם גאנצן דאקומענט דורך דער אפליקאציע, קען דאס פירן צו טייערערע LLM רופן און צו שוואכערע ענטפערס. contextual compression איז געמאכט צו פארריכטן דאס.

טריט:

- מען אימפארטירט ContextualCompressionRetriever.
- מען דעפינירט א פונקציע pretty_print_docs, וואס דרוקט די דאקומענטן שיין אויס, מיט דאפלטע נייע שורות צווישן יעדן page_content.
- מען וויקלט איין דעם vector store.
- מען שאפט א compressor מיט LLMChainExtractor.
- מען שאפט דעם ContextualCompressionRetriever מיט דעם compressor, און מיט אן base retriever פונעם vector store.
- מען שיקט אריין די פראגע.

אין דעם רעזולטאט זעט מען אז די דאקומענטן זענען קירצער ווי געווענליך, אבער עס זענען אויך פאראן איבערגעחזרטע זאכן.

אנדערע retrievers: TF-IDF און SVM (15:09:58)

עס ווערט באמערקט אז א vector DB איז נישט דאס איינציגע געצייג צו געפינען דאקומענטן. די retriever-אבסטראקציע פון LangChain נעמט אריין נאך שטייגערס, אזוי ווי TF-IDF אדער SVM.

מען לאדנט די environment variables, מען לאדנט א PDF , און מען צעטיילט דעם טעקסט מיט RecursiveCharacterTextSplitter. מען באשטימט chunk_size און chunk_overlap און מען צעטיילט.

דערנאך ניצט מען SVMRetriever און TFIDFRetriever. מען פרעגט "What are major topics for this class?" און מען זעט דעם page_content פונעם רעזולטאט. דערנאך לויפט מען די זעלבע זאך מיט דעם TF-IDF retriever.

דער קורס פארשלאגט אפצושטעלן און אליין פרובירן ביישפילן. אין דער קומענדיגער לעקציע ווערט געוויזן ווי אזוי מען פרעגט פראגעס און באקומט רעלעוואנטע ענטפערס.

אן אגענט איבער עטליכע דאקומענטן (15:10:31)

די ווייטערדיגע לעקציע ווייזט ווי אזוי מען פארברייטערט א געוועלכן אגענט צו באהאנדלען עטליכע דאקומענטן, מיט שטייגנדיגע מדרגות פון קאמפליצירטקייט.

ערשט שטעלט מען אויף דעם OpenAI API key, און מען אימפארטירט nest_asyncio. די ערשטע אויפגאבע איז אויפצושטעלן א function calling אגענט איבער דריי וויסנשאפטלעכע ארבעטן (papers). דאס טוט מען דורכדעם וואס מען שטעלט צוזאם די vector- און summary-געצייג פון יעדן דאקומענט אין איין רשימה, און מען גיט עס איבער צום אגענט, אזוי אז דער אגענט האט אינגאנצן זעקס געצייג.

מען האט אראפגעלאדנט דריי papers: MetaGPT, LongLoRA און Self-RAG.

מען פארוואנדלט יעדן paper אין א געצייג. דערפאר ניצט מען א הילפס-פונקציע get_doc_tools, וואס בויט אויטאמאטיש פאר א געגעבענעם paper סיי א vector index געצייג און סיי א summary index געצייג. דער vector געצייג טוט א vector זוך, און דער summary געצייג מאכט א סיכום פונעם גאנצן דאקומענט. פאר יעדן paper באקומט מען צוריק ביידע געצייג.

אלס LLM קלייבט מען GPT-3.5 Turbo פון OpenAI. אז מען ציילט די געצייג וואס ווערן איבערגעגעבן צום אגענט, קומט ארויס זעקס: דריי papers מיט צוויי געצייג פאר יעדן, א vector געצייג און א summary געצייג.

דער נעקסטער טריט איז צו בויען דעם אלגעמיינעם agent worker, וואס נעמט אריין אלע זעקס געצייג און דעם LLM. מיט אים קען מען פרעגן פראגעס איבער אלע דריי דאקומענטן צוזאמען, אדער אינערהאלב איין דאקומענט. אלס ערשטע פראגע הייבט מען אן צו פרעגן: "Tell me about the evaluation dataset".

פיל-דאקומענט אגענט, די צוקונפט פון RAG, און Agentic RAG (15:13:06)

ענדע פון דעם צוויי-פאפירן אגענט: LongLoRA און Self-RAG (15:13:06)

דער קורס ענדיגט דעם פריערדיגן ביישפיל, וואו דער אגענט אנטוואורט אויף פראגעס וועגן צוויי וויסנשאפטלעכע פאפירן. די פראגע איז געווען וועלכע דאטא-סעטס מען האט גענוצט אין LongLoRA, און וואס זענען געווען די רעזולטאטן פון דער אפשאצונג (evaluation). דער ענטפער איז געקומען אז איינער פון די eval דאטא-סעטס נוצט דעם test split, און מען האט געקענט זען די eval רעזולטאטן פאר די LongLoRA מאדעלן.

די נעקסטע פראגע איז געווען: "גיב מיר א סך-הכל פון ביידע, Self-RAG און LongLoRA ". דאס לאזט מאכן א summarization אויף ביידע פאפירן. צוערשט רופט מען דעם summary tool פאר Self-RAG, מיט דעם נאמען אלס input. ביים לויפן זעט מען שריט נאך שריט: ערשט "added user message to memory",

דערנאך "calling function" מיט דעם output, און נאכאמאל "calling function" מיט נאך א output, איינס פאר יעדן פאפיר.

צוגרייטן עלף פאפירן: URLs און טולס (15:13:50)

דערנאך ווערט געוויזן א גרעסערע ווערסיע מיט מער פאפירן. דער קאוד שטעלט אויף צוויי ליסטעס: urls און papers. די urls ליסטע האט 11 לינקס צו וויסנשאפטלעכע פאפירן אין PDF פארמאט פון דעם OpenReview וועבזייטל. די papers ליסטע האט די פאסנדע טעקע-נעמען, וואס שטעלן מסתמא פאר די אראפגעלאדענע ווערסיעס. יעדער URL אין urls איז פארבונדן מיט א געוויסן טעקע-נאמען אין papers, וואס ווייזט אז מען נוצט די נעמען כדי אפצוהיטן די אראפגעלאדענע טעקעס. אזא אויפשטעל איז ניצלעך כדי צו אויטאמאטיזירן דאס אראפלאדענען, נאכצופאלגן די טעקעס, אדער צו רופן יעדן פאפיר מיט זיין באשטימטן נאמען.

דער נעקסטער קאוד גייט דורך די ליסטע פאפירן און ציט ארויס פאר יעדן דאקומענט זיינע באזונדערע טולס מיט דער פונקציע get_doc_tools:

- מען אימפארטירט get_doc_tools פון דעם utils מאדול, און Path פון pathlib כדי צו באהאנדלען טעקע-נעמען.
- מען שאפט א ליידיגן dictionary מיטן נאמען paper_to_tools_dict, וואו מען וועט אפהיטן וועלכע טולס געהערן צו וועלכן פאפיר.
- מען גייט דורך די papers ליסטע, און פאר יעדן פאפיר דרוקט מען א מעלדונג וועלכער פאפיר ווערט יעצט באהאנדלט.
- פאר יעדן פאפיר רופט מען get_doc_tools און מען גיט איר דעם טעקע-נאמען און דעם נאמען אן דעם "pdf." סוף.
- די פונקציע גיט צוריק צוויי טולס: א vector tool , וואס פארוואנדלט מסתמא דעם טעקסט אין embeddings פאר NLP ארבעט, און א summary tool, וואס מאכט סך-הכלען פונעם פאפיר.
- די צוויי טולס ווערן אפגעהיטן אינעם dictionary, מיט דעם טעקע-נאמען אלס key און די טולס אלס value.

אזוי באקומט מען א שנעלע ארבעט מיט דאקומענטן, מיט גרינגן צוטריט סיי צו די וועקטאר-פארמען און סיי צו די סך-הכלען פון יעדן פאפיר. מען זעט אויך דעם output.

אלע טולס אין איין ליסטע, און אן אינדעקס איבער די טולס (15:15:57)

דער קאוד שאפט א ליסטע all_tools, וואס זאמלט צונויף די טולס פון paper_to_tools_dict פאר יעדן פאפיר אין papers. מען נוצט א nested list comprehension : מען גייט דורך יעדן פאפיר, נעמט זיינע צוויי טולס פון dictionary, און לייגט יעדן טול אריין אין all_tools. דער רעזולטאט איז איין פלאכע ליסטע מיט אלע vector און summary טולס פון אלע פאפירן, אנשטאט זיי האלטן געטיילט לויט פאפיר. אזוי איז גרינגער צו באהאנדלען אלע טולס צוזאמען.

דערנאך שאפט מען אן object-based אינדעקס איבער די טולס, מיט VectorStoreIndex פון דער llama_index.core ביבליאטעק. די פונקציע ObjectIndex.from_objects נעמט all_tools און ארגאניזירט זיי אין אן אינדעקס וואס מען קען זוכן אין אים. דער ארגומענט index_cls=VectorStoreIndex זאגט אז מען

זאל אינדעקסירן לויט וועקטארן, דאס הייסט יעדער טול ווערט פארוואנדלט אין א וועקטאר כדי מען זאל שנעל קענען זוכן. דאס לאזט ארויסנעמען די פאסנדע טולס לויט ענלעכקייט צו דעם באניצערס פראגע.

דער קאוד:

obj_index.as_retriever(similarity_top_k=3) = object_retriever

שאפט א retriever פונעם אינדעקס. די מעטאד as_retriever פארוואנדלט דעם אינדעקס אין א retriever וואס ברענגט רעזולטאטן לויט ענלעכקייט. similarity_top_k=3 זאגט אז ביי יעדער פראגע ברענגט ער די דריי ענלעכסטע רעזולטאטן. דאס איז ניצלעך אין זוך-מאשינען, chatbots אדער recommendation סיסטעמען.

ארויסנעמען טולס און קוקן אויף זייער metadata (15:17:59)

דער קאוד tools = (...)object_retriever.retrieve בעט דעם retriever צו געפינען וואס פאסט צו דער פראגע. ער זוכט אינעם אינדעקס און ברענגט די פאסנדסטע רעזולטאטן לויט ענלעכקייט, ביז דריי שטיק, ווייל אזוי איז ער אויפגעשטעלט. די רעזולטאטן ווערן אפגעהיטן אינעם tools variable, און מען קען זיי ווייזן דעם באניצער אדער ווייטער געבן פאר דעם AI מאדעל. אזוי ארבעט מען אפט אין AI זוך-סיסטעמען, chatbots און פראגע-ענטפער אפליקאציעס.

tools[2].metadata נעמט ארויס די metadata פונעם דריטן רעזולטאט. יעדער רעזולטאט האט סיי דעם תוכן וואס מען האט געפונען און סיי metadata , וואס פלעגט אנטהאלטן מקור, צייט-שטעמפל, מחבר, דאקומענט ID און נאך פרטים. דאס הילפט צו פארשטיין פון וואנעט דער רעזולטאט קומט.

אויפשטעלן דעם אגענט און שטעלן פראגעס (15:19:32)

יעצט שטעלט מען אויף אן AI אגענט וואס ענטפערט פראגעס לויט דער אינפארמאציע וואס ער נעמט ארויס פון אלע פאפירן:

- מען אימפארטירט FunctionCallingAgentWorker און AgentRunner פון llama_index.core.agent.
- מען שאפט אן agent worker מיט FunctionCallingAgentWorker , און מען גיט אים דעם object_retriever אלס דעם מכשיר צו געפינען אינפארמאציע, און דעם LLM כדי צו פארארבעטן און מאכן ענטפערס.
- דער system prompt זאגט דעם אגענט אז ער זאל זיך אלעמאל פארלאזן אויף דער אינפארמאציע וואס ער האט געפונען, און נישט אויף זיין פריערדיגער וויסן.
- צולעצט: AgentRunner(agent_worker) = agent. דאס נעמט אריין דעם worker אין AgentRunner, וואס מאכט גרינגער צו פירן און לויפן דעם אגענט'ס ארבעט.

אזא אגענט איז ניצלעך פאר אנאליז פון וויסנשאפטלעכע דאקומענטן און פאר פראגע-ענטפער אפליקאציעס.

די ערשטע פראגע: וועלכער evaluation דאטא-סעט ווערט געניצט אין MetaGPT, און ווי פארגלייכט ער זיך מיט SWE-bench. די query פונקציע שיקט די פראגע צום אגענט, וואס נעמט ארויס די פאסנדע אינפארמאציע דורכן object_retriever און פארארבעט עס מיטן שפראך-מאדעל. דער ענטפער ווערט אפגעהיטן אין response, און ((print(str(response פארוואנדלט אים אין טעקסט און ווייזט אים. אזוי באקומט מען א גוט-אויסגעשטעלטן פארגלייך צווישן די צוויי דאטא-סעטס, לויט די דאקומענטן.

די לעצטע פראגע: אנאליזירן און פארגלייכן צוויי פאפירן, LongLoRA און LoftQ. די פראגע בעט דעם אגענט ערשט צו אנאליזירן דעם צוגאנג פון יעדן פאפיר, און דערנאך זיי צו פארגלייכן און וויזן די אונטערשיידן. דער אגענט נעמט ארויס אינפארמאציע מיטן object_retriever, פאראלבעט מיטן שפראך-מאדעל, און דער ענטפער ווערט אפגעהיטן אין response . דאס איז ניצלעך פאר אקאדעמישע פארשונג, טעכנישע אנאליז און AI סך-הכלען. מען זעט דעם output.

סיום: די צוקונפט פון RAG (15:21:52)

דער RAG טייל ענדיגט מיט א בליק אויף דער צוקונפט. אזוי שטארק ווי RAG איז היינט, זעט די צוקונפט אויס נאך מער פארשפרעכנדיג. אין די קומענדיגע יארן קען מען ערווארטן:

- קלוגערע retrievers: געוויסערע retrievers וואס נעמען אין באטראכט דעם טיפערן טייטש, דעם באציערס חשקות און כוונה, נישט נאר שליסל-ווערטער.
- Multimodal RAG: נישט נאר טעקסט, נאר אויך בילדער, קול און ווידעא אין דעם ארויסנעמען און שאפן.
- פערזענליכע אגענטן: סיסטעמען וואס לערנען זיך פון דעם באציערס feedback אין רעאלער צייט און געבן מער פאסנדע און פערזענליכע ענטפערס.
- Memory-augmented RAG: א לאנג-טערמיניגער זכרון אין אגענטן, כדי זיי זאלן געדענקען דעם קאנטעקסט פון איין סעסיע צו דער אנדערער.
- Low-latency און scalable RAG: פארבעסערונגען אין אפהיטן און ארויסנעמען, כדי RAG זאל גלאט ארבעטן אפילו אין ריזיגע מאסן און מיט גרויס גיכקייט.

RAG אנטוויקלט זיך שנעל: פון זוך-מאשינען ביז קונה-שטיצע, געזונט, פינאנצן און חינוך. וועו די סיסטעמען ווערן מער צוגעפאסט און געבויט אויף געוויסער, רעאלער וויסן, וועלן זיי פארבינדן סטאטישע דאטא מיט לעבעדיגע, קלוגע ענטפערס, און עפענען דעם וועג פאר דעם קומענדיגן דור AI.

מען פאסט איבער וואס מען האט געלערנט: וואס RAG איז, פארוואס עס איז וויכטיג, ווי עס טרייבט רעאלע אפליקאציעס; די הויפט-טיילן: query encoder, retriever, reranker, knowledge base און generator; און prakt־ישע demos: לאדענען דאקומענטן, chunking, אפהיטן מיט embeddings און vector stores, און בויען א פיל-דאקומענט אגענט.

וואס איז אן אגענט? (15:24:05)

דערנאך הייבט זיך אן א נייער טייל, און מען גייט ערשט ערקלערן וואס אן אגענט איז. אין AI איז אן אגענט א סיסטעם וואס נעמט אליין וואר זיין סביבה, באשליסט לויט דער דאטא, און טוט פעולות כדי צו דערגרייכן באשטימטע צילן. אן אגענט קען האנדלען מיט זיין סביבה, זיך לערנען פון איר, און צופאסן זיין אויפפירונג. אין מער קאמפליצירטע סיסטעמען קענען אגענטן באשליסן, למשל אנאליזירן מוסטערן, פאראויסזאגן, אדער אליין דורכפירן ארבעט לויט דעם קאנטעקסט און דעם באציערס אינטעראקציע.

ביישפיל: א virtual assistant איז אן אגענט. וועו מען פרעגט אים א פראגע, נעמט ער וואר די סביבה, דאס הייסט די פראגע, ער פאראלבעט די דאטא, און גיט א ענטפער לויט זיין knowledge base. ער קען אויך אליין באשליסן, למשל שיקן דערמאנונגען אדער איינשטעלן מיטינגס לויט דעם באציערס חשקות און קאלענדאר.

וואס איז RAG? (15:25:07)

RAG, retrieval-augmented generation, איז אן AI טעכניק וואס פאראייניגט צוויי צוגאנגען כדי צו פארבעסערן די ענטפערס:

- Retrieval: דער סיסטעם נעמט ארויס די פאסנדע דאטא פון אן אויסערליכער knowledge base, דאטאבאזע אדער זאמלונג דאקומענטן.
- Generation: דער generative מאדעל פאראַרבעט די דאטא כדי צו שאפן א געוויין ענטפער וואס פאסט צום קאנטעקסט.

RAG סיסטעמען פארגרעסערן וואס generative מאדעלן קענען, דורך ברענגען לעבעדיגע דאטא פון אויסערליכע מקורות אין רעאלער צייט. אזוי ווערן די ענטפערס מער אינפארמירט, געוויי און פאסיג.

ביישפיל: א chatbot פאר קונה-שטיצע. אויב ער פארלאזט זיך נאר אויף זיין אינערליכן וויסן, איז ער באגרענעצט צו וואס מען האט אים ביים טריינירן געגעבן. מיט RAG קען דער chatbot פרעגן א דערהיינטיגטע knowledge base און ענטפערן מיט דער נייסטער אינפארמאציע, למשל פראדוקט-דערהיינטיגונגען, צוריקגעב-כללים אדער וויפיל סחורה איז פאראן.

וואס איז Agentic RAG? (15:26:08)

יעצט פאראייניגט מען ביידע געדאנקען: אגענטישע אויפפירונג און RAG. אן agentic RAG סיסטעם איז אן AI וואס נעמט נישט נאר ארויס אינפארמאציע פון אויסערליכע מקורות און שאפט ענטפערס, נאר טוט אויך פעולות לויט דער אינפארמאציע און לויט די באשלוסן וואס ער מאכט אליין. אזעלכע סיסטעמען גייען ווייטער ווי פשוט ענטפערן פראגעס: זיי טוען ארבעט, צינדן אן געשעענישן, און האנדלען לויט דער ארויסגענומענער דאטא. זיי זענען געבויט צו זיין מער זעלבסטשטענדיג, און נוצן אויספולן די כח סיי פון ארויסנעמען דאטא און סיי פון באשליסן.

ביישפיל: א פראיעקט-פארוואלטונג געהילף וואס ארבעט מיט אן אגענט. מען בעט אים צו געפינען די נעקסטע פרייַ צייט פאר א מיטינג פון דער מאנשאפט. דער געהילף נעמט ארויס די דאטא פון דעם קאלענדאר, קוקט אויב עס זענען דא קאנפליקטן, און שטעלט אליין איין דעם מיטינג לויט ווען די באטייליגטע זענען פריי, אן דעם וואס דער באניצער זאל דארפן אריינמישן. דערנאך גייט דער קורס איבער צו דער פראגע פארוואס מען דארף agentic RAG.

אגענטישע RAG מיט LlamaIndex און דער Router Query Engine (15:27:40)

פארוואס מען ניצט אגענטישע RAG סיסטעמען (15:27:40)

דער קורס ערקלערט אז אגענטישע RAG סיסטעמען זענען ווערטפול, ווייל זיי גיבן א פולשטענדיגע לייזונג פאר קאמפליצירטע אויפגאבעס, וואס קען וואקסן מיט די באדערפענישן. זיי פאראייניגן צוויי מעלות: אויטאנאמע באשלוס-מאכן, און באווייטערן דעם ענטפער מיט דרויסנדיגע דאטא. עס ווערן אויסגערעכנט פיר הויפט-סיבות:

- אויטאנאמע באשלוס-מאכן: די סיסטעמען קענען מאכן באשלוסן וואס זענען באזירט אויף דרויסנדיגע דאטא. אזוי קען מען אויטאמאטיזירן פראצעסן וואס האבן אמאל געדארפט א מענטש וואס זאל קוקן דערויף, למשל צוטיילן ווען צו ענטפערן אויף אימעילס, אדער פארטיילן ארבעט.

• ענטפערס וואס פארשטייען דעם קאנטעקסט: ווייל די סיסטעם ברענגט אינפארמאציע פון דער זעלבער רגע, קען זי שאפן ענטפערס וואס זענען שטארק שייך צום ענין, געווי, פאסיג צום קאנטעקסט, און אקטועל.

• אויטאמאטיזירן אויפגאבעס: די סיסטעמען קענען דורכפירן פארשידענע ארבעטן, ווי שיקן אימעילס, איינשטעלן מיטינגס, אדער דערהיינטיגן רעקארדס, אלעס לויט דער אינפארמאציע וואס זיי האבן אויפגעזוכט. דאס רעדוצירט די נויטוועניקייט פון א מענטש זיך אריינצומישן.

• סקאלאביליטעט: ווייל זיי קענען נאכאנאנד ארויסברענגען דרויסנדיגע דאטא און זי ניצן פאר באשלוסן און אקציעס, קען מען זיי לייכט אויסשפרייטן אויף אסאך באניץ-פעלער און סביבות.

וואס איז LlamaIndex, און פארוואס ניצט מען עס (15:28:58)

אין דער לעקציע ווערט ערקלערט אז דער יסוד פון בויען אגענטישע RAG אפליקאציעס איז די מעגליכקייט צו ברענגען דאטא אויף אן עפעקטיווע אופן, און דערמיט באווייטערן דעם גענעראטיוון פראצעס. דא קומט אריין LlamaIndex. פריער האט עס געהייסן GPT Index. עס איז א פריימווערק וואס מאכט גרינגער דעם פראצעס פון שאפן RAG אפליקאציעס. עס דערלויבט פראגראמירער צו אינדעקסירן גרויסע מאסן דאטא פון פארשידענע מקורות, ווי דאקומענטן, דאטאבעיסעס אדער API-ס. דערנאך ניצט מען די דאטא צו באווייטערן געזעראטיוע מאדעלן ווי GPT. מיט LlamaIndex קען מען אויף אן עפעקטיווע אופן פירן דאס ארויסברענגען פון דאטא, מאכן די מאדעלן מער אינפארמירט, און דערמעגליכן דער אגענטישער RAG סיסטעם צו האנדלען אליין לויט אינפארמאציע פון דער זעלבער רגע.

פארוואס ניצט מען LlamaIndex? פיר סיבות:

• עפעקטיוו אינדעקסירן דאטא: עס דערלויבט צו ארגאניזירן און אינדעקסירן די דאטא, אזוי אז מען קען לייכט ארויסנעמען די שייכותדיגע אינפארמאציע לויט די פראגעס אדער די באדערפענישן פון דער סיסטעם.

• גלאטע אינטעגראציע: עס פארבינדט זיך מיט געזעראטיוע מאדעלן ווי GPT, און דערמעגליכט צו ברענגען דאטא פון דער זעלבער רגע כדי צו פארבעסערן די ענטפערס.

• מען קען עס צופאסן: עס איז בייגעווידיג, און דערלויבט צו באשטימען ווי אזוי די דאטא ווערט אינדעקסירט און ארויסגענומען, אזוי אז עס פאסט פאר די ספעציפישע באדערפענישן פון דער אפליקאציע.

• סקאלאביל: עס קען האנדלען מיט גרויסע דאטא-סעטס. דאס איז קריטיש פאר אפליקאציעס וואס דארפן וואקסן.

פינף טריט צו בויען אן אגענטישע RAG אפליקאציע מיט LlamaIndex (15:30:45)

דער קורס שטעלט אוועק די הויפט-טריט. זיי פירן דורך דעם גאנצן פראצעס: איינשטעלן די סביבה, אינדעקסירן די דאטא, און פארבינדן אלעס מיט אן אגענט וואס קען אליין מאכן באשלוסן.

טריט 1: איינשטעלן די אנטוויקלונג-סביבה. מען דארף א סביבה מיט די נויטיגע געצייג און לייברעריס:

• אינסטאלירן Python, ווייל LlamaIndex איז געבויט אויף Python.

• אינסטאלירן די דעפענדענסיס: מיט א פעקל-פארוואלטער ווי pip אינסטאלירט מען LlamaIndex און אנדערע נויטיגע לייברעריס, ווי OpenAI, אויב מען ניצט מאדעלן באזירט אויף GPT.

• איינשטעלן די API שליסלען: וועו מען ניצט דרויסנדיגע מאדעלן ווי OpenAI, דארף מען איינשטעלן די נויטיגע API keys כדי צו באקומען צוטריט.

טריט 2: אינדעקסירן די דאטא וואס די סיסטעם וועט ארויסברענגען. די דאטא קען קומען פון אינערליכע דאקומענטן, פון דאטאבעיסעס אדער פון דרויסנדיגע API-ס:

- לאדן די דאטא: מען קלייבט צוזאם די דאטא און מען לאדט זי אריין אין א פאסיגן פארמאט, צום ביישפיל טעקסט-דאקומענטן, CSV טעקעס, אדער דאטא פון אן API.
- שאפן אן אינדעקס: מיט LlamaIndex פראצעסירט מען די דאטא און מען שאפט א סטרוקטור וואס מען קען לייכט פרעגן.
- אפהיטן דעם אינדעקס: נאכן אינדעקסירן, היט מען עס אפ אויף אן אופן וואס מען קען לייכט צוקומען צו אים ביי צוקונפטיגע פראגעס.

טריט 3: אויפשטעלן די לאגיק פון ארויסברענגען. דער ציל איז אז די סיסטעם זאל ברענגען די שייכותדיגע אינפארמאציע, לויט דעם וואס דער באניצער שרייבט אריין אדער לויט פראגעס פון דער סיסטעם אליין:

- באשטימען די זוך-פראגעס: מען שטעלט אויף לאגיק וואס פראצעסירט אנקומענדיגע פראגעס, און ברענגט די שייכותדיגע דאטא פונעם אינדעקסירטן וויסן-באזע.
- צופאסן די פאראמעטערס: מען פיין-טונט דעם פראצעס דורך באשטימען וואס הייסט "שייכותדיגע דאטא". צום ביישפיל, מען שטעלט איין שוועלן (thresholds) פאר ענלעכקייט אדער שייכות.

טריט 4: בויען די אגענטישע לאגיק. דא באשטימט מען ווי אזוי די סיסטעם וועט מאכן באשלוסן און דורכפירן אקציעס:

- באשלוס-מאכן: מען שרייבט לאגיק וואס לאזט די סיסטעם באשליסן וועלכע אקציעס צו נעמען לויט דער אינפארמאציע. דער ביישפיל: אויב די סיסטעם געפינט אז א באניצער דארף א מיטינג, זאל זי אליין קענען קוקן אינעם קאלענדאר, געפינען פרייע צייטן, און איינשטעלן דעם מיטינג.
- באהאנדלען אקציעס: מען פארבינדט דרויסנדיגע סערוויסעס, ווי קאלענדאר API-ס אדער אימעיל-סערוויסעס, אזוי אז די סיסטעם זאל קענען שיקן מעסעדזשעס, איינשטעלן געשעענישן, אדער אפילו שאפן באריכטן.

טריט 5: טעסטן און אינטעגרירן. צום סוף פאראייניגט מען דאס ארויסברענגען פון דאטא, די גענעראטיוע מאדעלן, און די אגענטישע באשלוס-לאגיק אין איין פונקציאנירנדיגע סיסטעם. דערנאך טעסט מען:

- טעסטן דאס ארויסברענגען: זיכער מאכן אז די סיסטעם ברענגט די ריכטיגע דאטא אויף די פראגעס פון די באניצער.
- טעסטן די אגענטישע אקציעס: באשטעטיגן אז דער אגענט נעמט פאסיגע אקציעס לויט דער דאטא, ווי איינשטעלן אויפגאבעס, שיקן נאטיפיקאציעס, אדער אנצינדן אנדערע פראצעסן.

די קאמפאנענטן פון אגענטישע RAG (15:35:25)

עס ווארן אויסגערעכנט זעקס קאמפאנענטן:

-) Knowledge baseוויסן-באזע): דער דרויסנדיגער דאטא-מקור, פון וואנען דער אגענט ברענגט אינפארמאציע.
- דער אגענט: ער איז פאראנטוואורטליך צו ברענגען דאטא, זי צו פראצעסירן, און צו מאכן אויטאנאמע באשלוסן.

• דער מעכאניזם פון ארויסברענגען: ער ברענגט די שייכותדיגע דאטא פון דער וויסן-באזע לויט די פראגעס.

• דער גענעראטיוואר מאדעל: ער פראצעסירט די ארויסגעבראכטע דאטא, און שאפט ענטפערס וואס זענען געניו לויטן קאנטעקסט.

• די אקציע-לאגיק: זי באשטימט וועלכע אויפגאבעס דער אגענט קען דורכפירן אליין, לויט דער דאטא.

• דער Router Query Engine: ער שיקט די פראגעס צום פאסיגן אגענט אדער קאמפאנענט, כדי זיי זאלן ווערן פראצעסירט אויף אן עפעקטיווע אופן.

ווי אזוי ווערט א פראגע פראצעסירט אין דער סיסטעם (15:36:26)

מען זעט א פארפשוטערטן פלוס, ווי אזוי די קאמפאנענטן ארבעטן צוזאמען:

• באקומען די פראגע: די סיסטעם באקומט א פראגע פונעם באניצער. דאס קען זיין עפעס פשוט'ס ווי "וואן איז מיין נעקסטער מיטינג?", אדער א מער קאמפליצירטע בקשה ווי "שטעל איין א מיטינג מיט John מארגן 10:00 פארמיטאג".

• רוטירן די פראגע: דער Router Query Engine אנאליזירט די פראגע, און באשטימט וועלכער אגענט אדער קאמפאנענט פאסט אם בעסטן פאר איר. צום ביישפיל, ער קען זי שיקן צום אגענט וואס איז אחראי אויף איינשטעלן צייטן.

• ארויסברענגען דאטא: דער אגענט וואס האט באקומען די ארבעט, אנצינדט דעם מעכאניזם וואס ברענגט די שייכותדיגע דאטא פון דער וויסן-באזע. אין דעם פאל קען די סיסטעם ציען די קאלענדאר-דאטא, אדער קוקן צי John איז פריי.

• פראצעסירן די דאטא: דער גענעראטיוואר מאדעל ניצט די דאטא צו שאפן א ענטפער וואס איז געניו לויטן קאנטעקסט. ער קען אויך שאפן א באשרייבונג פון דער אויפגאבע, למשל "איינשטעלן א מיטינג".

• דורכפירן די אקציע: די אקציע-לאגיק נעמט איבער, און דער אגענט שטעלט אליין איין דעם מיטינג, אדער נעמט אן אנדערע נויטיגע אקציע.

• פארענדיגן די אויפגאבע: די סיסטעם באשטעטיגט די אקציע, למשל: "דיין מיטינג מיט John איז איינגעשטעלט פאר 10:00 פארמיטאג מארגן", און גיט איבער דעם רעזולטאט צום באניצער.

וואס איז א ראוטער? דער משל פונעם לופטפעלד (15:38:14)

דער נעקסטער קאפיטל באהאנדלט ברייטער דעם Router Query Engine. אין דער וועלט פון AI און נאטירליכע שפראך-פראצעסירונג, איז ער איינער פון די וויכטיגסטע קאמפאנענטן פון אן עפעקטיווע סיסטעם. פריער ווערט ערקלערט וואס א ראוטער איז, מיט א משל: שטעל דיר פאר אז דו ביסט אין א גרויסן לופטפעלד און דו דארפסט כאפן א פלי. ס'גייען אסאך פליגערס צו פארשידענע ערטער, און דו מוזט וויסן צו וועלכן טויער (gate) צו גיין. דער ראוטער אין אן AI סיסטעם ארבעט ענלעך. ער וויזט די פראגעס, בקשות און פראמפטס פון די באניצער צו דער ריכטיגער סיסטעם, וואס זאל זיי פראצעסירן.

געניער געראדט, טוט א ראוטער דריי זאכן:

• ער הערט די פראגע אדער די בקשה פונעם באניצער.

• ער פאנטאזירט אויס דעם בעסטן וועג ווי צו באהאנדלען די פראגע: צי מען דארף אויפזוכן פאקטן, צי מען דארף שאפן א קרעאטיוון ענטפער, אדער עפעס צווישן ביידע.

• ער שיקט די פראגע צו דער פאסיגער סיסטעם כדי צו באקומען א ענטפער.

מען קען זיך פארשטעלן דעם ראוטער ווי א פארקער-קאנטראלירער, וואס זארגט אז יעדע פראגע גייט צום ריכטיגן ארט.

וואס איז א Router Query Engine, און פארוואס איז ער וויכטיג (15:39:30)

דער Router Query Engine איז א סיסטעם וואס ניצט דעם ראוטער צו פארוואלטן און באהאנדלען די פראגעס פון די באניצער אויף אן עפעקטיווע אופן. ער איז ווי דער מוח הינטער דעם פארקער-קאנטראלירער: ער באשליסט וואוהין צו שיקן יעדע פראגע, לויט דער נאטור פון דער בקשה. מער טעכניש געזעדט, איז ער א שליסל-קאמפאנענט אינעם אגענטישן RAG פריימווערק. RAG פאראייניגט צוויי טעכנאלאגיעס: א סיסטעם וואס ברענגט אינפארמאציע (retrieval), און א גענעראטיווע סיסטעם. דער Router Query Engine באשליסט צי א פראגע זאל גיין צו דער ארויסברענג-סיסטעם, כדי צו באקומען פאקטישע ענטפערס פון א דאטאבעיס; צו א גענעראטיוון מאדעל, כדי צו שאפן קרעאטיוון אינהאלט ווי מעשיות אדער סיכומים; אדער צו א קאמבינאציע פון ביידע.

פארוואס איז ער אזוי וויכטיג? פיר סיבות:

• עפעקטיווע פארוואלטונג פון פראגעס: אן א ראוטער וואלט די סיסטעם פראבירט צו פראצעסירן אלע סארטן פראגעס אויפן זעלבן אופן. צום ביישפיל: "וואס איז די הויפטשטאט פון ?France" פאדערט א פשוטן פאקטישן ענטפער, דעריבער דארף די פראגע גיין צו דער ארויסברענג-סיסטעם. אבער "שרייב א ליד וועגן דעם ים" פאדערט קרעאטיווע שאפונג, דעריבער דארף זי גיין צום גענעראטיוון מאדעל.

• פארבעסערן די פערפארמענס: ווייל יעדע פראגע ווערט פראצעסירט אויפן מערסט עפעקטיוון אופן, ווערט אויסגעמיטן איבעריגע ארבעט, און די ענטפערס קומען שנעלער.

• מער גענויקייט: אז מען שיקט די פראגעס צו דער ריכטיגער סיסטעם, זענען די ענטפערס מער גענוי און שייכותדיג. מען שיקט נישט קיין קרעאטיווע אויפגאבעס צו א פאקטישער סיסטעם, און פארקערט.

• סקאלאביליטעט: ווען די סיסטעם וואקסט, און מען לייגט צו נייע סארטן פראגעס אדער נייע מאדעלן, קען מען דעם Router Query Engine לייכט צופאסן צו באהאנדלען זיי, אן פיל איבער-קאנפיגורירן.

ווי אזוי ארבעט דער Router Query Engine, שריט ביי שריט (15:41:52)

דער קורס צעטיילט דעם פראצעס אויף פשוטע טריט:

• ערשטער טריט, באקומען די פראגע: דער Router Query Engine באקומט די פראגע אדער דעם פראמפט פונעם באניצער. דאס קען זיין, למשל: "וואס איז ?quantum computing", "שרייב א ליד וועגן דעם ים", אדער "מאך א סיכום פון דער לעצטער AI פארשונג".

• צווייטער טריט, אויסקלאסיפיצירן די כוונה: נאכן באקומען די פראגע, קלאסיפיצירט זי דער ראוטער, כדי צו באשטימען וואספארא ענטפער מען דארף. דער פראצעס אידענטיפיצירט צי די פראגע פאדערט פאקטישע אינפארמאציע (דעפיניציעס, דאטעס אדער פאקטן), א סיכום (פון אן ארטיקל אדער א פארשונג), אדער גענעראטיווע ענטפערס (קרעאטיוו שרייבן, מיינונגען, אדער קאמפליצירטע ערקלערונגען).

• דריטער טריט, רוטירן צו דער ריכטיגער סיסטעם: נאכן קלאסיפיצירן, באשליסט דער ראוטער וואוהין צו שיקן די בקשה. די ערקלערונג פון די פארשידענע ריכטונגען גייט ווייטער אינעם קומענדיגן חלק.

דער Router Query Engine אין פראקטיק און Tool Calling (15:43:03)

וויאזוי דער ראוטער שיקט יעדע פראגע צום ריכטיגן סיסטעם (15:43:03)

דער קורס האלט ביי ערקלערן די שריט פון א Router Query Engine. ביים דריטן שריט, וועו דער ראוטער באשליסט וואוהין צו שיקן די פראגע, זענען דא עטליכע מעגלעכקייטן:

- א Factual Retriever: אויב די פראגע איז א פאקטישע, למשל "וואס איז די הויפטשטאט פון פראנקרייך?", שיקט דער ראוטער זי צו א knowledge-based retriever, וואס זוכט דעם ריכטיגן ענטפער אין א דאטאבאזע.
- א געזעראטיוועו מאדעל: אויב די פראגע פאדערט שעפערישע אדער פרייע שאפונג, למשל "שרייב א ליד וועגן דעם ים", שיקט דער ראוטער זי צו א געזעראטיוון מאדעל ווי GPT אדער אנדערע AI מאדעלן וואס קענען שאפן אינהאלט.
- א געמישטער (hybrid) צוגאנג: א פראגע ווי "דערצייל מיר וויאזוי AI ווערט געזונטהייט" פאדערט אמאל סיי פאקטן סיי שעפערישקייט. דער ראוטער קען דעמאלט שיקן איין טייל פון דער פראגע צו א retriever און דעם אנדערן טייל צו א געזעראטיוון מאדעל.

דער פערטער שריט איז צוריקגעבן דעם ענטפער: נאכדעם וואס די פראגע איז אנגעקומען צום ריכטיגן סיסטעם, באארבעט דאס סיסטעם די פראגע און שאפט א ענטפער, און דער ראוטער שיקט דעם ענטפער צוריק צום באניצער.

עס ווערט געבראכט א ביישפיל: ביי דער פאקטישער פראגע "וואס איז די הויפטשטאט פון פראנקרייך" דערקענט דער ראוטער אז דאס איז א פאקטישע פראגע און שיקט זי צום knowledge-based retriever. יענער זוכט אויס דעם ענטפער און גיט צוריק: "די הויפטשטאט פון פראנקרייך איז פאריז."

צוגרייטן דעם API שליסל און די ביבליאטעקן (15:44:38)

דערנאך ווערט אלעס גענויזן אין קאוד. די ערשטע שורה ברענגט אריין א פונקציע get_openai_api_key פון א מאדול וואס הייסט helper. די נעקסטע שורה רופט די פונקציע און היט אפ דעם OpenAI API שליסל אין א ווערי‌אבל, כדי דער פראגראם זאל אים שפעטער קענען נוצן. אין קורצן: דער קאוד שטעלט צו דעם שליסל וואס מען דארף כדי צו ארבעטן מיט OpenAI'ס סערוויסעס.

דער נעקסטער קאוד ברענגט אריין די ביבליאטעק nest_asyncio. די פונקציע nest_asyncio.apply() דערמעגלעכט צו לויפן אסינכראנישן קאוד אין סביבות ווי Jupyter Notebooks, וואו עס לויפט שוין אן event loop. זי דערלויבט "איינגעשאכטלטע" event loops, אזוי אז מען קען לויפן async פונקציעס אן קאנפליקטן.

אריינלאדן דעם PDF און צעטיילן אין שטיקער (15:45:10)

שפעטער ברענגט דער קאוד אריין די קלאס SimpleDirectoryReader פונעם core מאדול פון דער ביבליאטעק LlamaIndex. די קלאס ליינט און לאדט דאקומענטן פון א געוויסן דירעקטאריע אדער פון א סעט טעקעס. די שורה זעט אויס אזוי:

```
documents = SimpleDirectoryReader(input_files=["metagpt.pdf"]).load_data()
```

דער טייל מיט input_files שאפט אן אינסטאנץ פון דער קלאס און שטעלט אן אז די טעקע metagpt.pdf איז די אריינגאנג-טעקע, דאס הייסט אז דער ליינער וועט לאדן דאטא פון דעם PDF. די מעטאד load_data()

ליינט און לאדט די דאטא פון דער טעקע, און דער רעזולטאט ווערט אפגעהיט אינעם ווערייאבל documents, וואס אנטהאלט יעצט דעם אינהאלט פון דער MetaGPT טעקע, גרייט פאר ווייטערדיגע באארבעטונג. פשוט געזאגט: מען לאדט א PDF אריין אין פראגראם כדי מען זאל קענען נוצן זיין אינהאלט.

דערנאך ברענגט מען אריין SentenceSplitter פון llama_index.core.node_parser, מען היט אים אפ אין א ווערייאבל splitter מיט א chunk_size פון 1024, און מען צעטיילט די דאקומענטן אין) nodes שטיקער) דורך splitter.get_nodes_from_documents(documents).

אויסשטעלן די מאדעלן: GPT-3.5 Turbo און Embeddings (15:47:24)

דער נעקסטער קאוד ברענגט אריין די נויטיגע טיילן פון LlamaIndex כדי צו קאנפיגורירן מאדעלן פאר טעקסט-שאפונג און embeddings. די קלאס Settings ווערט געוונצט צו קאנפיגורירן דאס סיסטעם; די קלאס OpenAI באהאנדלט טעקסט-שאפונג מיטן מאדעל GPT-3.5 Turbo; און די קלאס OpenAIEmbedding שאפט embeddings מיטן מאדעל text-embedding-ada-002.

- Settings.llm = OpenAI(model="gpt-3.5-turbo") שטעלט אויס דעם שפראך-מאדעל וואס שאפט מענטשן-ענלעכן טעקסט.
- Settings.embed_model = OpenAIEmbedding(model="text-embedding-ada-002") שטעלט אויס דאס מאדעל וואס שאפט נומעריש פארשטעלונגען פון טעקסט, וואס ווערן געוופן embeddings.

אזוי איז דאס סיסטעם גרייט סיי פאר טעקסט-שאפונג סיי פאר סעמאנטישער אנאליז.

צוויי אינדעקסן און צוויי Query Engines (15:48:26)

דער קאוד שאפט צוויי סארטן אינדעקסן מיט LlamaIndex:

- SummaryIndex: ארגאניזירט די) nodesדי דאטא-שטיקער) פאר עפעקטיוו סיכומים, כדי מען זאל לייכט קענען ארויסנעמען קורצע אינפארמאציע פון א זאמלונג דאטא.
- VectorStoreIndex: היט אפ די nodes ווי וועקטאר-פארשטעלונגען (embeddings), וואס דערמעגלעכט זוכן לויט ענלעכקייט אין סעמאנטישן באדייט.

ביידע זענען געבויט פאר שנעלן ארויסנעמען דאטא: דער ערשטער פאר סיכומים, דער צווייטער פאר פראגעס לויט ענלעכקייט.

דערנאך שאפט מען צוויי query engines. דער summary_query_engine פארוואנדלט דעם SummaryIndex אין א query engine וואס גיט צוריק סיכומים אין א בוים-סטרוקטור (tree summarize) און באארבעט פראגעס אסינכראניש פאר שנעלערער פערפארמענס. דער vector_query_engine פארוואנדלט דעם VectorStoreIndex אין א query engine וואס טוט פראגעס לויט ענלעכקייט, מיט די וועקטאר-פארשטעלונגען פון דער דאטא.

די כלים (Tools) און זייערע באשרייבונגען (15:49:28)

דער קאוד שאפט צוויי QueryEngineTool אביעקטן, יעדער פאר אן אנדער צוועק:

- summary_tool: ניצט דעם summary_query_engine צו באהאנדלען פראגעס וואס פאדערן סיכומים וועגן MetaGPT, וועו באניצער פרעגן אויף אלגעמיינע אינפארמאציע אדער איבערבליקן. די באשרייבונג

וואס ווערט געגעבן איז: "Useful for summarization questions related to MetaGPT", וואס מאכט קלאר אז דער כלי איז ספעציעל געבויט פאר סיכומים.

- vector_tool: ניצט דעם vector_query_engine צו ארויסנעמען ספעציפישן אינהאלט פונעם MetaGPT ארטיקל. דורך זוכן לויט וועקטאר-ענלעכקייט געפינט ער רעלעוואנטע אינפארמאציע לויטן סעמאנטישן באדייט פון דער באניצער'ס פראגע. די באשרייבונג: "Useful for retrieving specific context from the MetaGPT paper".

די כלים זענען וויכטיג כדי צו געבן צילגעריכטע ענטפערס: אדער א סיכום פון גרויסע שטיקער דאטא, אדער ספעציפישע רעלעוואנטע אינפארמאציע, לויט וואס די פראגע פאדערט.

בויען דעם Router Query Engine (15:51:00)

יעצט שאפט מען דעם RouterQueryEngine וואס שיקט פראגעס צום פאסיגן כלי לויטן סארט פראגע:

- selector=LLMSingleSelector.from_defaults(): דער סעלעקטאר העלפט דעם ראוטער באשליסן וועלכן כלי צו נוצן.
- query_engine_tools=[summary_tool, vector_tool]: די צוויי כלים וואוהין דער ראוטער קען שיקן: דער ערשטער פאר סיכומים, דער צווייטער פאר ארויסנעמען קאנטעקסט לויט וועקטאר-ענלעכקייט.
- verbose=True: געבט אויספירלעכע לאגס כדי מען זאל קענען נאכפאלגן דעם ראוטינג-פראצעס און ווי די פראגע ווערט באהאנדלט, פאר דורכזיכטיגקייט.

די ערשטע פראגע: א סיכום פונעם דאקומענט (15:52:00)

די ערשטע פראגע: query_engine.query("What is the summary of the document?"). דער ראוטער וועט די פראגע שיקן צום פאסיגן כלי; ווי‌באלד עס איז א סיכום-פראגע, וועט זי מסתמא גיין צום summary_tool. נאכן לויפן דרוקט מען אויס print(str(response)), וואס פארוואנדלט דעם ענטפער אין א סטרינג (אויב ער איז נאך נישט) און ווייזט דעם סיכום פונעם דאקומענט אויפן קאנסאל.

דערנאך דרוקט מען אויס len(response.source_nodes), און דער רעזולטאט איז 34. source_nodes זענען די טיילן פונעם דאקומענט וואס זענען געווארן צו שאפן דעם ענטפער, און len רעכנט אויס וויפיל nodes) שטיקער) האבן בייגעשטייערט צום ענטפער.

נאך פראגעס: אגענטן און די Ablation Study (15:53:33)

די צווייטע פראגע: "How do agents share information with other agents?". דער query engine באארבעט זי ענלעך און שיקט זי, לויט דער קאנפיגוראציע, צום vector_tool אדער summary_tool, וואס ניצט די רעלעוואנטע אינפארמאציע צו שאפן א ענטפער, און מען דרוקט אויס דעם ענטפער. ביים לויפן ווייזט דער לאג: "Selecting query engine 0", מיטן טעם אז די ברירה איז מער רעלעוואנט ווייל זי באציט זיך צו סיכום-פראגעס וועגן MetaGPT; ערשט באארבעט עס, דערנאך גיט עס דעם רעזולטאט.

צום סוף שטעלט מען אלעס צוזאמען, לויפט נאכאמאל דעם פריערדיגן קאוד, און שיקט די פראגע "Tell me about the ablation study results". דער query engine שיקט די פראגע צום פאסיגן כלי, נעמט ארויס די רעלעוואנטע אינפארמאציע פון metagpt.pdf און היט אפ דעם רעזולטאט אין response, און print(str(response)) ווייזט דעם ענטפער. דער ענטפער: די ablation study רעזולטאטן ווייזן אז MetaGPT

באהאנדלט עפעקטיוו די שווערייקייטן וואס האבן צו טאן מיט ניצן דעם קאנטעקסט. דערמיט ענדיגט זיך די לעקציע.

א נייע לעקציע: וואס איז Tool Calling? (15:55:39)

די נעקסטע לעקציע הייבט אן מיט Tool Calling. מען וועט זען וויאזוי אן AI אגענט וואס ניצט א שפראך-מאדעל קען נישט נאר אויסקלייבן א פונקציע צו לויפן, נאר אויך אויסרעכענען די ארגומענטן וואס די פונקציע דארף. דאס איז א טייל פונעם גרעסערן פראצעס פון tool calling און agentic RAG (retrieval augmented generation) אפליקאציעס, וואס פארגרעסערט שטארק וואס AI קען.

Tool Calling מיינט די פעאיגקייט פונעם אגענט צו רופן דינאמיש דרויסנדיגע כלים, ווי APIs, דאטאבאזעס אדער ספעציעלע פונקציעס, כדי ארויסצונעמען, צו באארבעטן און צו שאפן ענטפערס. דאס איז קריטיש אין RAG סיסטעמען, ווען דער אגענט דארף אפט צוטריט צו לעבעדיגע (real-time) דאטא אדער סטרוקטורירטע אינפארמאציע וואס געפינט זיך נישט אין דער דאטא מיט וועלכער דאס מאדעל איז טרייניירט געווארן.

פארוואס איז Tool Calling אזוי וויכטיג? (15:56:40)

ביי א געווענליכן מאדעל איז די וויסן פון AI סטאטיש, באזירט אויף דער דאטא מיט וועלכער ער איז טרייניירט געווארן. Tool Calling לייגט צו א שיכט פון דינאמיק: דער אגענט קען ברענגען אקטועלע אינפארמאציע, דורכפירן חשבונות און ארבעטן מיט דרויסנדיגע סיסטעמען כדי צו ענטפערן קאמפליצירטע פראגעס. אזוי ווערט דער אגענט מער אנפאסיג און קען ענטפערן פינקטלעך אפילו ווען די דאטא איז נישט פארהאן אין זיין ארגינעלן טרייניינג-סעט.

אין RAG אפליקאציעס גייט tool calling געווענליך לויט א סטרוקטורירטער רייע שריט, וואו דער AI אגענט באשליסט ווען און ווי צו ברענגען דרויסנדיגע אינפארמאציע און לויפן ספעציפישע פונקציעס. דער ערשטער שריט איז באארבעטן די באניצער'ס פראגע: דער פראצעס הייבט זיך אן ווען דער באניצער שיקט אריין א פראגע.

RAG מיט טול-רופן: טעאריע און א דעמא (15:57:42)

ווי אזוי דער אגענט באשליסט צי ער דארף דרויסנדיגע דאטא (15:57:42)

דער קורס ערקלערט ווייטער די שריט פון א RAG סיסטעם וואס ארבעט מיט טול-רופן. דער ערשטער שריט איז אנאליזירן די פראגע: דער אגענט קוקט אויף דער פראגע און באשליסט צי מען קען זי ענטפערן מיט דעם אינערליכן וויסן וואס דער מאדעל פארמאגט שוין, אדער מען דארף ברענגען דאטא פון דרויסן.

ווען די פראגע פאדערט אינפארמאציע פון יעצט, למשל דאס וועטער היינט אדער די פרייזן פון סטאקס, אדער ווען זי פאדערט סטרוקטורירטע דאטא, למשל SQL פראגעס אדער API רופן, גרייט זיך דער אגענט צו רופן די דרויסנדיגע טולס. א ביישפיל: א פראגע ווי "וואס איז די לעצטע נייעס ווענן ?AI" פאדערט א טול וואס רופט אן API און ברענגט די לעבעדיגע נייעס.

ארויסנעמען און פארשטארקן (Retrieval and Augmentation) (15:58:15)

ווען דער סיסטעם האט באשלאסן אז מען דארף דרויסנדיגע דאטא, קלייבט ער אויס דעם פאסיגן וועג ווי צו ברענגען די דאטא. די געווענליכע וועגן זענען:

- סעמאנטישע זוך מיט וועקטאר-דאטאבאזעס ווי Pinecone, FAISS אדער Weaviate. דארט ווערן דאקומענטן געזוכט לויט זייער באדייט, נישט נאר לויט די ענויע וועגטער.
- SQL אדער NoSQL פראגעס, כדי צו ברענגען סטרוקטורירטע דאטא פון דאטאבאזעס.
- API רופן, כדי צו ברענגען אינפארמאציע פון יעצט, למשל וועטער-דאטא פון OpenWeather אדער סטאק-פרייזן דורך פינאנציעלע APIs.

דער שריט לייגט צו צום ענטפער פונעם מאדעל די נייסטע און מערסט שייכותדיגע אינפארמאציע, און דאדורך ווערט דער ענטפער מער גענוי.

באארבעטן און באשטעטיגן די דאטא (15:58:46)

נאכדעם וואס מען האט ארויסגעברענגט די דאטא, מוז דער סיסטעם זיך פארזיכערן אז די אינפארמאציע איז ריכטיג און שייך איידער ער ניצט זי אין ענטפער. אויב מען האט ארויסגעברענגט עטליכע דאקומענטן, שטעלט דער סיסטעם זיי אין א סדר לויט ווי שייך זיי זענען, לויט זייער באדייט. ער קען אויך קירצן גרויסע מאסן דאטא, און ארויסנעמען דופליקאטן אדער אינפארמאציע וואס סותר'ט זיך.

אויב די פראגע פאדערט א חשבון, למשל "וויפיל איז 10% פון 2,000 דאלער?", קען דער סיסטעם לויפן א Python פונקציע וואס מאכט דעם חשבון און דעם רעזולטאט אריינלייגן אין ענטפער. דער שריט זארגט אז די דאטא וואס ווערט גענוצט אין לעצטן ענטפער איז גילטיג און נוצליך.

שאפן דעם לעצטן ענטפער (15:59:47)

וועו די אינפארמאציע איז באשטעטיגט, שאפט דער סיסטעם דעם לעצטן ענטפער. דער גרויסער שפראך-מאדעל (LLM) פאראייניגט דעם אגענט'ס אינערליכן וויסן מיט דער דאטא וואס ער האט ארויסגעברענגט, און שרייבט א ענטפער וואס איז קלאר, האט א סדר און איז קורץ. דער ענטפער ווערט ארויסגעגעבן אין א פארעם וואס איז גרינג צו פארשטיין פארן באניצער. אויב עס איז דא אינפארמאציע וואס סותר'ט זיך, קען דער אגענט ברענגען עטליכע שיטות, אדער אנווייזן אויף די מקורות.

פארוואס טול-רופן מאכט RAG שטארקער (16:00:18)

עס ווערט ערקלערט אז טול-רופן גיט דעם מאדעל א לעבעדיגן כח, ווייל ער קען צוקומען צו דאטא פון יעצט:

- ער קען רופן APIs און ברענגען די לעצטע אינפארמאציע, ווי סטאק-פרייזן אדער נייעס.
- ער קען אויספירן א גאנצע ארבעט, למשל מאכן חשבונות, לויפן פראגעס, אדער ארבעטן מיט דרויסנדיגע דאטאבאזעס.
- ער קען באשטעטיגן און פארבעסערן: דער אגענט קען פארגלייכן עטליכע מקורות פון דאטא און ארויסנעמען איבעריגע אינפארמאציע, אזוי אז דער לעצטער ענטפער איז גענוי.

דאס מאכט דעם RAG סיסטעם מער זעלבשטענדיג, און ער קען האנדלען מיט קאמפליצירטע פראגעס וואס פאדערן אינפארמאציע פון יעצט, בפרט אין געביטן ווי ביזנעס-אינטעליגענץ, קונדן-סערוויס און פארשונג.

הקדמה צו דער דעמא (16:00:48)

אין דער לעקציע ווערט געוויזן א דעמא ווי אזוי א RAG סיסטעם מיט טול-רופן ארבעט אין דער פראקטיק. די דעמא ווייזט ווי דער אגענט ניצט דרויסנדיגע טולס, ווי APIs אדער א דאטאבאזע, כדי צו ברענגען דאטא וואס איז נישט טייל פון זיין אינערליכן וויסן. זי ווייזט אויך ווי די ארויסגעברענגטע דאטא ווערט גענוצט צו

פארבעסערן דעם ענטפער, אז ער זאל זיין מער גענוי און לעבעדיג. מען זעט ווי דער אגענט רופט טולס לויט דעם סארט פראגע: צי עס איז אינפארמאציע פון יעצט, סטרוקטורירטע דאטא, אדער א געוויסע פונקציע ווי א חשבון.

אריינברענגען דעם OpenAI שליסל און nest_asyncio (16:01:49)

דער ערשטער שורה קאוד ברענגט אריין פון א מאדול וואס הייסט helper . מען נעמט אן אז דאס איז א Python טעקע, מסתמא helper.py, מיט פארשידענע הילפס-פונקציעס. די פונקציע get_openai_api_key איז געמאכט צו ברענגען דעם OpenAI API שליסל. דער שליסל איז נויטיג כדי צו באשטעטיגן ווער עס שיקט די בקשות צו די OpenAI סערוויסעס, ווי די GPT מאדעלן. אז מען ברענגט אריין די פונקציע, קען מען זי רופן און באקומען דעם שליסל, אן אים שרייבן גלייך אין קאוד.

דערנאך רופט דער קאוד get_openai_api_key און לייגט דעם רעזולטאט אריין אין דעם ווערייאבל OPENAI_API_KEY. די פונקציע נעמט מסתמא דעם שליסל פון א זיכערן מקור, ווי א סביבה-ווערייאבל (environment variable), א קאנפיגוראציע-טעקע, אדער א סיסטעם פאר סודות. אזוי קען מען ניצן דעם שליסל איבערן גאנצן קאוד ביי אלע OpenAI ארבעטן. דער קורס מאנט אז יעדער זאל זיך באזארגן מיט אן אייגענעם שליסל.

דערנאך ווערט אריינגעברענגט די ביבליאטעק nest_asyncio. דאס איז א Python פעקל וואס לאזט ניצן asyncio, Python'ס איינגעבויטע ביבליאטעק פאר קאוד וואס לויפט נישט אויף איינמאל, אויך אין סביבות וואס לויפן שוין זייער אייגענעם "event loop", ווי Jupyter notebooks. אן דעם לאזט asyncio נישט שטענדיג לויפן איין event loop אינעווייניג פון אן אנדערן. אין Jupyter איז דאס א פראבלעם, ווייל דארט לויפט שוין א event loop פאר די אינטעראקציעס. nest_asyncio גיט א וועג ארום דעם, אז אזעלכע לופס איינס אין צווייטן זאלן ארבעטן אן טעותים.

דער קאוד רופט nest_asyncio.apply(). די apply מעטאד ענדערט ווי דער asyncio event loop פירט זיך: זי לאטעט דעם עקזיסטירנדיגן לופ אז ער זאל שטיצן לופס איינס אין צווייטן. אזוי קען מען לויפן עטליכע אסינכראנישע ארבעטן צוגלייך, אן קאנפליקטן און אן טעותים.

דעפינירן פשוטע טולס מיט FunctionTool (16:04:25)

FunctionTool איז א קלאס פון דער LlamaIndex פריימווארק. זי וויקלט איין געווענליכע Python פונקציעס און מאכט פון זיי טולס וואס מען קען אריינשטעלן אין א גרעסערן סיסטעם, למשל פאר באארבעטן דאקומענטן, פראגעס, אדער אנדערע ארבעטן אין LlamaIndex. אזוי ווערט א נארמאלע Python פונקציע א טול וואס מען קען ניצן אינעם ארבעטס-גאנג פון אן אגענט.

מען דעפינירט צוויי פונקציעס:

- add: נעמט צוויי גאנצע צאלן (x, integers) און y, לייגט זיי צוזאמען און גיט צוריק די סומע. די type annotations ווייזן קלאר אז זי נעמט צוויי integers און גיט צוריק איין integer.
- mystery: נעמט אויך צוויי integers, x און y, אבער זי לייגט זיי קודם צוזאמען, און דערנאך מאכט זי דעם רעזולטאט אין קוואדראט, דאס הייסט זי מערט די סומע מיט זיך אליין. די פונקציע ווייזט ווי מען וויקלט איין מער קאמפליצירטע אפעראציעס אלס טולס.

דערנאך:

FunctionTool.from_defaults(fn=add) = add_tool

FunctionTool.from_defaults(fn=mystery) = mystery_tool

די מעטאד from_defaults שטעלט איין דעם טול מיט די נארמאלע איינשטעלונגען און פארבינדט אים מיט דער פונקציע. ביידע זענען יעצט אביעקטן פון FunctionTool. מען קען זיי אריינשטעלן אין LlamaIndex אגענטן אדער אנדערע סיסטעמען, אזוי אז די פונקציעס זענען מאדולאר און מען קען זיי ניצן נאכאמאל.

דער מאדעל רופט דעם טול (16:06:28)

מען ברענגט אריין די OpenAI קלאס אינערהאלב LlamaIndex. דאס לאזט דעם סקריפט ניצן OpenAI'ס שפראך-מאדעלן כדי צו באארבעטן דאטא און שאפן ענטפערס אויף א סדר'דיגן אופן, למשל פאר שאפן טעקסט, קירצן, אדער ענטפערן אויף פראגעס. דערנאך שאפט מען א מאדעל-אביעקט: llm = OpenAI(model="gpt-3.5-turbo"). עס ווערט ערקלערט אז GPT-3.5 Turbo איז א שטארקע ווערסיע פון GPT-3.5, באקאנט אלס יעפעקטיוו און נישט טייער.

דערנאך רופט מען די מעטאד predict_and_call אויפן llm אביעקט. מען גיט איר א רשימה טולס, [add_tool, mystery_tool], און די פראגע "Tell me the output of the mystery function on 2 and 9", און verbose=True. די מעטאד לאזט דעם LLM ניצן די געגעבענע טולס כדי צו אויספירן די ארבעט. דער אגענט באארבעט די בקשה, קלייבט דעם mystery טול און רעכנט אויס די פונקציע אויף 2 און 9. verbose=True שאלטעט איין א פרטימדיגן לאג, כדי מען זאל קענען נאכפאלגן די אינערליכע שריט פונעם אגענט. צום סוף דרוקט מען אויס דעם ענטפער נאכן אים פארוואנדלען אין א סטרינג. מען זעט אז דער רעזולטאט איז 121, דאס הייסט (9+2) אין קוואדראט.

לאדן א PDF דאקומענט (16:08:33)

דער קאוד ברענגט אריין SimpleDirectoryReader פון llama_index.core. די קלאס לאדט און באארבעט דאקומענטן פון טעקעס אדער פאלדערס, ווי PDF, טעקסט-טעקעס און אנדערע פארמאטן וואס זענען געשטיצט, און גרייט זיי צו פאר וויטערדיגע באארבעטונג אדער אינדעקסירן. מען ניצט זיי שפעטער פאר ארבעטן ווי טעקסט-אנאליז אדער ארויסנעמען אינפארמאציע.

documents = SimpleDirectoryReader(input_files=["metagpt.pdf"]).load_data()

דער פאראמעטער input_files זאגט וועלכע טעקעס מען זאל לאדן. דא לאדט מען איין טעקע, metagpt.pdf, פון דעם ארבעטס-פאלדער. מען קען אויך אנגעבן עטליכע טעקעס אדער א גאנצן פאלדער, און דער רידער וועט זיי לאדן. די מעטאד load_data לייענט דעם אינהאלט פון די טעקעס, נעמט ארויס דעם טעקסט פונעם MetaGPT דאקומענט און גרייט אים צו. דער רעזולטאט ווערט אריינגעלייגט אין דעם וועריאבל documents.

צעטיילן אין שטיקלעך מיט SentenceSplitter (16:10:05)

SentenceSplitter איז א קלאס אין LlamaIndex וואס צעטיילט גרויסע דאקומענטן אין קלענערע שטיקלעך (chunks), אפט לויט די זאצן. דאס איז נוצליך פאר אינדעקסירן, ווייל קלענערע שטיקלעך מאכן די פראגעס און די באארבעטונג פון דאטא מער יעפעקטיוו.

splitter = SentenceSplitter(chunk_size=1024). דער פאראמעטער chunk_size באשטימט די מאקסימאלע גרייס פון יעדן שטיקל, געמאסטן אין טאקענס. דא וועט יעדער שטיקל אנטהאלטן ביז 1024 טאקענס. דאס שטימט מיט א געווייסע צאל ווערטער אדער זאצן, לויט ווי דער שפראך-מאדעל צעטיילט

טעקסט אין טאקענס. אזוי צעטיילט דער splitter דעם דאקומענט אין קלענערע אפטיילן וואס מען קען גרינגער באארבעטן, און קיין שטיקל גייט נישט אריבער די באשטימטע גרייס. דער וועריאבל nodes וועט אנטהאלטן די שטיקלעך וואס קומען ארויס דערפון.

נאדעס, וועקטאר-אינדעקס און אגענט-ריזאנירן (16:11:37)

די נאדעס און זייער מעטאדאטא (16:11:37)

דער קורס הייבט אן דא בשעת מען האלט שוין ביי די נאדעס (nodes). דער דאקומענט איז צעטיילט געווארן אין קלענערע שטיקלעך (chunks). די שטיקלעך קען מען ווייטער באניצן, למשל צו בויען א דאקומענט-אינדעקס אדער צו ענטפערן אויף פראגעס אויף א שנעלן אופן.

עס ווערט ערקלערט אז די נאדעס פארבעסערן די פערפארמענס. די סיסטעם ארבעט מיט קליינע שטיקלעך דאטא, און זי דארף נישט פראצעסירן א גרויסן דאקומענט מיטאמאל.

דערנאך ווערט געוויזן א קאוד וואס קוקט אריין אין דעם ערשטן נאד:

- nodes[0] נעמט ארויס דאס ערשטע שטיקל טעקסט פון דער ליסטע נאדעס.
- מען רופט אן get_content מיט metadata_mode=all. דאס ברענגט דעם גאנצן אינהאלט פונעם נאד צוזאמען מיט אלע מעטאדאטא, למשל דער מקור (source), דער צייט-שטעמפל (timestamp) און אנדערע פרטים.
- דער רעזולטאט ווערט ארויסגעדרוקט אויפן קאנסאל.

וועז מען לויפט דעם קאוד, זעט מען דעם page label און דעם טעקסט פונעם preprint, מיט אלע מעטאדאטא.

VectorStoreIndex און דער query engine (16:12:08)

דערנאך ערקלערט דער קורס וואס איז VectorStoreIndex. דאס איז א קלאס אינעם LlamaIndex פריימווערק וואס שאפט אן אינדעקס פון וועקטארן. דער קלאס מאכט עס מעגליך צו האלטן, ארויסנעמען און פרעגן (query) דאקומענט-דאטא וואס איז אויסגעדריקט ווי וועקטארן, און דאס אלעס אויף אן עפעקטיוון אופן. דערפאר איז ער גוט צו זוכן אדער ארויסנעמען אינפארמאציע לויט ווי ענליך די וועקטארן זענען איינער צום אנדערן.

מען שאפט דעם אינדעקס ענליך צו דעם:

VectorStoreIndex(nodes) = vector_index

די שורה שאפט א VectorStoreIndex אביעקט און גיט אים אריין די נאדעס, הייסט די קליינע טעקסט-שטיקלעך פונעם פריערדיקן שריט. יעדער נאד ווערט פארוואנדלט אין א וועקטאר דורך מעטאדן ווי embedding מאדעלן. אזא מאדעל שטעלט דעם טעקסט פאר מיט נומערן, און אזוי קען מען שנעל זוכן ענליכקייטן. דער אינדעקס האלט יעצט די וועקטארן, אז מען זאל זיי קענען גיך דערגרייכן וועז מען פרעגט.

דערנאך קומט:

vector_index.as_query_engine(similarity_top_k=2)

די מעטאד מאכט פונעם וועקטאר-אינדעקס א query engine, א מאשין וואס מען קען פרעגן און זי ברענגט ארויס אינפארמאציע. דער פאראמעטער similarity_top_k=2 זאגט אז בייי יעדער פראגע ברענגט די מאשין צוריק די צוויי וועקטארן וואס זענען די מערסטע ענליך צו דער פראגע. אזוי קען מען געפינען די נאדעס, הייסט די טעקסט-שטיקלעך, וואס זענען די נאנטסטע צו א געוויסער פראגע.

פילטערן לויט מעטאדאטא (16:13:40)

יעצט ווערט אריינגעברענגט MetadataFilters. דאס איז א קלאס פון LlamaIndex וואס לאזט פילטערן די רעזולטאטן לויט די מעטאדאטא וואס איז פארבונדן מיט די דאקומענטן אדער מיט די נאדעס אינעם אינדעקס. מיט אים קען מען שארפער מאכן די רעזולטאטן, ווייל מען באגרעניצט די זוכעניש צו געוויסע מעטאדאטא-ווערטן.

דער query engine ווערט דאס מאל געשאפן מיט צוויי פאראמעטערס:

- similarity_top_k=2 ברענגט צוריק די צוויי מערסטע ענליכע רעזולטאטן, לויט דער וועקטאר-ענליכקייט צווישן דער פראגע און די נאדעס.
- filters=MetadataFilters.from_dicts(...) לייגט צו א פילטער לויט מעטאדאטא. די מעטאד from_dicts נעמט א ליסטע פון דיקשענערי'ס (dictionaries), און יעדער איינער האט א key און א value. אין דעם פאל זוכט דער פילטער נאר נאדעס וואס זייער page_label איז "2".

מיט די סעטינגס געפינט דער query engine די מערסטע ענליכע וועקטארן, און ער מאכט אויך זיכער אז די רעזולטאטן זענען נאר פון נאדעס מיט page_label = 2.

דערנאך שיקט מען אריין די פראגע:

response = query_engine.query("What are some high-level results of MetaGPT?")

די מאשין זוכט די נאדעס וואס זענען די מערסטע ענליך צו דעם טעקסט פון דער פראגע, און זי האלט זיך דערביי צום מעטאדאטא-פילטער, הייסט נאר נאדעס פון בלאט 2. דער רעזולטאט ווערט געהאלטן אין response. דערנאך דרוקט מען אים ארויס מיט print(str(response)). די str פונקציע פארוואנדלט דעם response אביעקט אין א סטרינג, ווי מען האט פריער אויסגעשמועסט. מען זעט אז דער ענטפער ווערט ארויסגעדרוקט.

די source nodes (16:15:43)

אין response.source_nodes געפינט זיך די ליסטע פון די נאדעס פון וועלכע מען האט געבויט דעם ענטפער. יעדער נאד איז א שטיקל דאטא אדער א דאקומענט וואס האט מיטגעהאלפן צום ענטפער פונעם query engine.

מען לויפט איבער זיי מיט א לופ:

for n in response.source_nodes:

print(n.metadata)

יעדער n האט אין זיך סיי דעם אינהאלט און סיי די מעטאדאטא וואס האבן געדינט צו מאכן דעם לעצטן ענטפער. פאר יעדן נאד דרוקט מען ארויס זיין מעטאדאטא. די מעטאדאטא קען אנטהאלטן צוגאב-אינפארמאציע, למשל דעם טיטל פונעם מקור-דאקומענט, דעם מחבר, דעם page label אדער אנדערע

אייגנשאפטן וואס מען האט צוגעלייגט צום נאד. וועו מען דרוקט ארויס די מעטאדאטא, פארשטייט מען פון וואו די דאטא איז געקומען און אין וועלכן קאנטעקסט.

א פונקציע vector_query מיט בלאט-נומערן (16:16:46)

דערנאך ווייזט דער קורס א נייעם שטיקל קאוד און ערקלערט אים שריט נאך שריט:

- List ווערט באניצט צו זאגן אז דער פאראמעטער page_numbers איז א ליסטע פון סטרינגס. אזוי קען די פונקציע באקומען עטליכע בלאט-נומערן אויף איינמאל.
- FilterCondition איז אן enumeration פון LlamaIndex. זי באשטימט ווי אזוי מען לייגט צוואם עטליכע פילטער-באדינגונגען וועו מען פרעגט דעם אינדעקס. ביי AND מוזן אלע באדינגונגען זיין אמת. ביי OR איז גענוג אז כאטש איינע זאל זיין אמת.
- די פונקציע vector_query טוט א וועקטאר-זוכעניש איבערן אינדעקס, לויט א טעקסט-פראגע און לויט א ליסטע בלאט-נומערן וואס מען מעג אבער מוז נישט געבן. דער פאראמעטער query איז די פראגע אלס סטרינג. מען מאכט פון איר אן embedding און מען באניצט עס צו געפינען ענליכקייט צווישן וועקטארן. דער פאראמעטער page_numbers איז א ליסטע בלאט-נומערן צו פילטערן די רעזולטאטן לויט געוויסע בלעטער. אויב די ליסטע איז ליידיג אדער מען האט זי נישט געגעבן, זוכט די פונקציע איבער אלע בלעטער.
- metadata_dicts ווערט געשאפן מיט א list comprehension. דאס מאכט א ליסטע פון דיקשענערי'ס, און יעדער איינער איז א key-value פאר פאר א מעטאדאטא-פילטער. דער key איז page_label , און דער value איז יעדער נומער פון דער ליסטע page_numbers.
- מען רופט אן vector_index.as_query_engine מיט similarity_top_k=2, ווי פריער, אז ער זאל צוריקברענגען די צוויי מערסטע ענליכע רעזולטאטן.
- filters=MetadataFilters.from_dicts(metadata_dicts, condition=FilterCondition.OR) שאפט די פילטערס פון די דיקשענערי'ס. דער FilterCondition.OR מאכט זיכער אז דער query engine ברענגט צוריק רעזולטאטן וואו כאטש איינער פון די page labels שטימט מיט איינעם פון די געגעבענע בלאט-נומערן.
- צום סוף נעמט FunctionTool.from_defaults די פונקציע vector_query און וויקלט זי איין אין א FunctionTool. אזוי קען מען זי באניצן אין א גרעסערן LlamaIndex ווארקפלאו.

דאס LLM רופט אן דעם געצייג (16:19:50)

יעצט שאפט דער קאוד אן OpenAI מאדעל דורכן קלאס OpenAI, מיטן מאדעל GPT-3.5 Turbo און temperature = 0. די סעטינג מאכט אז דער מאדעל גיט דעטערמיניסטישע ענטפערס, הייסט יעדעס מאל דער זעלבער ענטפער. דאס איז גוט וועו מען דארף קאנסיסטענטע און פינקטלעכע רעזולטאטן, בפרט ביי אויפגאבעס וואו מען דארף געניע אינפארמאציע.

דערנאך רופט מען אן predict_and_call אויפן LLM אביעקט. מען גיט אריין דעם vector_query טול און א פראגע וועגן די הויפט-רעזולטאטן (high-level results) פון MetaGPT אויף בלאט 2. דער ארגומענט verbose=True ווייזט א דעטאלירטן לאג, און אזוי זעט מען ווי די פראגע ווערט דורכגעפירט. דער ענטפער ווערט געהאלטן אינעם response פאר ווייטערדיקן באנוץ.

דער ענטפער זאגט אז MetaGPT דערגרייכט א נייעם state-of-the-art אויף קאוד-געניראציע בענטשמארקס, מיט 85.9% און 87.7% אין Pass@1.

דערנאך ווערט נאכאמאל ערקלערט אז response.source_nodes איז די ליסטע נאדעס וואס האבן מיטגעהאלפן צום לעצטן ענטפער, און יעדער נאד איז א שטיקל דאטא וואס איז געווען א טייל פון דער זוכעניש. דער response אביעקט האלט דעם רעזולטאט פון דער פראגע. דער רעזולטאט קען אנטהאלטן טעקסט, מעטאדאטא אדער אנדערע אינפארמאציע וואס קומט פון די דאקומענטן אינעם אינדעקס. וועז מען לויפט עס, זעט מען דעם page label, file name, file path און די איבעריגע מעטאדאטא.

א צווייטער טול: SummaryIndex (16:21:21)

יעצט לייגט מען צו נאך א טול. מען אימפארטירט SummaryIndex און QueryEngineTool פון llama_index.core. מען גיט אריין די נאדעס אין SummaryIndex, און מען שאפט א query engine מיט response_mode="tree_summarize" און use_async=True. דער טול באקומט א באשרייבונג וואס זאגט אז ער איז נוצלעך וועז מען וויל א סאמערי פון MetaGPT.

דערנאך שטעלט מען א פראגע מיט verbose=True: "What are the MetaGPT comparisons with ChatDev described on page 8?" די function output זאגט אז MetaGPT איז בעסער ווי ChatDev אויפן דאטא-סעט פון סאפטווער-אנטוויקלונג, לויט פארשידענע מעסטונגען (metrics). אזוי האט דער סיסטעם געגעבן דעם ענטפער אויף דער פראגע.

מען דרוקט ארויס נאכאמאל די מעטאדאטא. דער page_label איז 8. מען האט געפרעגט וועגן בלאט 8, הייסט דער ענטפער שטימט, ווייל ער איז נאר פון page_label 8. מען זעט אויך דעם file name, וואס איז metagpt, דעם file path און אלעס וואס מען דארף.

צום לעצט פרעגט מען עפעס אנדערש: "What is the summary of the paper?" דער ענטפער וואס קומט ארויס זאגט אז "דער ארטיקל רעדט נישט וועגן די השפעה פון קלימאט-ענדערונג אויף ביאדייווערסיטעט און עקאסיסטעמען". מען זעט אז דער סיסטעם האט נישט געקענט געבן א סאמערי פונעם ארטיקל. דער קורס זאגט אז מען וועט דאס פארריכטן אין דער קומענדיקער לעקציע און דארט געפינען דעם ריכטיקן ענטפער.

אגענט-ריזאנירן אין שריט (16:22:56)

עס ווערט ערקלערט מיט פשוטע ווערטער וואס מען לערנט יעצט: ווי אזוי צו באהאנדלען קאמפליצירטע פראגעס וואס פארלאנגען עטליכע שריט. ביז יעצט, וועז מען האט געפרעגט א פראגע, האט דער סיסטעם אויסגעקליבן דעם ריכטיקן טול און געגעבן דעם ענטפער מיט איין מאל. אבער אויב די פראגע איז קאמפליצירטער און האט עטליכע טיילן, דארף מען האבן א מער ארגאניזירטן פראצעס וואס צעטיילט זי אין שריט וואס מען קען באהערשן.

דער קורס צעטיילט דעם ענין אין עטליכע אונטער-טעמעס, און ער גייט אריין אין פרטים ביי יעדער איינער:

- פארשטיין אגענט reasoning loops
- function calling אגענטן
- באהאנדלען קאמפליצירטע פראגעס
- בויען אינטעליגענטע אגענטן

אן agent reasoning loop איז א פראצעס אין וועלכן אן אינטעליגענטע סיסטעם באהאנדלט א קאמפליצירטע אויפגאבע אדער פראגע. זי צעטיילט זי אין קלענערע לאגישע אויפגאבעס. אנשטאט לייזן דעם גאנצן פראבלעם מיט איין מאל, פראצעסירט די סיסטעם איין שריט נאכן אנדערן, און זי פאסט צו איר אקציעס לויט וואס איז ארויסגעקומען פון די פריערדיקע שריט. אזוי איז זי פיל בעסער ביי פארפלאנטערטע פראבלעמען מיט אסאך זייטן, וואס מען קען נישט לייזן אין איין single forward pass.

די שריט אינעם reasoning loop:

- Input reception: די סיסטעם באקומט צוערשט די פראגע אדער די אויפגאבע.
- Problem breakdown: די סיסטעם אנאליזירט די פראגע, זי דערקענט די פארשידענע טיילן פונעם פראבלעם, און זי ארגאניזירט זיי אין שטיקלעך אדער שריט וואס מען קען באהערשן.
- Action selection: די סיסטעם באשליסט וועלכע אקציע אדער וועלכן טול צו באניצן פאר יעדן שטיקל. דאס קען זיין אנרופן א געוויסע פונקציע, צוזאמקלייבן דאטא, אדער ארויסברענגען א שלוס (inference).
- Processing and evaluation: די סיסטעם פירט דורך די אויסגעקליבענע אקציעס און אפשאצט די רעזולטאטן.
- Reiteration: לויט די רעזולטאטן קען די סיסטעם זיך אומקערן צו פריערדיקע שריט, צו פארבעסערן אדער צוצופאסן די לייזונג.

וואס איז א function calling אגענט (16:25:00)

א function calling אגענט איז אן אגענט וואס איז געבויט צו ארבעטן גלייך מיט פארשידענע פונקציעס אדער טולס. אזעלכע אגענטן טוען נישט נאר אריבערשיקן די פראגע צום ריכטיקן טול. זיי רופן אן פונקציעס צו טון ספעציפישע אויפגאבעס, אלס טייל פון לייזן א גרעסערן פראבלעם. אזוי קען די סיסטעם צוואמשטעלן עטליכע פראצעסן, און יעדע פונקציע ווערט באניצט אויפן עפעקטיוסטן אופן פאר איר אונטער-אויפגאבע. דערנאך הייבט דער קורס אן צו ערקלערן ווי אזוי עס ארבעט.

פונקציע-רופנדיקע אגענטן און קאמפליצירטע פראגעס (16:26:00)

פונקציע-רופן און זייערע מעלות (16:26:00)

דער קורס ערקלערט ווייטער וועגן "function calling" אגענטן. אנשטאט אן אלגעמיינעם פראצעס פון אויסקלייבן א געצייג, קען דער אגענט רופן ספעציפישע פונקציעס אויף א סטרוקטורירטן אופן. למשל: ער רופט איין פונקציע צו ברענגען דאטא, נאכדעם א צווייטע פונקציע צו פראצעסירן די דאטא, און דערנאך א דריטע פונקציע צו וויזן די דאטא בילדלעך (visualize).

דערנאך קומט דער באגריף פון א "סדר פון אקציעס": יעדער פונקציע-רוף איז א טייל פון א געפלאנטן סדר פון שריט וואס פירט צום ענדגילטיגן ענטפער. דאס מאכט דעם אגענט אסאך מער בייגעוודיג, און ער קען באהאנדלען ארבעטן וואס פאדערן קאמפליצירטע שריט.

די מעלות פון פונקציע-רופנדיקע אגענטן:

- עפעקטיווקייט: די זעלבע פונקציעס קען מען ווידער ניצן ביי פארשידענע ארבעטן, וואס מאכט דעם אגענט מער עפעקטיוו.

- מאדולאריטעט: מען קען צעטיילן ארבעטן אין קלענערע, קלאר-דעפינירטע פונקציעס, און דאס מאכט די סיסטעם גרינגער צו אויפהאלטן און מער בייגעוודיג.
- וואוקסיגקייט (scalability): דער אגענט קען רופן מער קאמפליצירטע אדער נאך פונקציעס לויט ווי נויטיג, אן צו שטערן די גאנצע סיסטעם.

וואס מאכט א פראגע קאמפליצירט (16:27:02)

א קאמפליצירטע פראגע איז אזא וואס פאדערט עטליכע שריט, נעמט דאטא פון פארשידענע קוואלן, אדער האט באזונדערע טיילן. א ביישפיל: "וויפיל איז געווען די פירמע'ס איינקונפט פארגאנגען יאר, און ווי פארגלייכט עס זיך מיט דעם זעלבן קוואַרטאל אין די פריערדיגע יארן?" אזא פראגע פאדערט צוזאמנעמען היסטארישע דאטא, מאכן חשבונות און פארגלייכן די רעזולטאטן. א קאמפליצירטע פראגע האט אפט באדינגונגען און אפהענגיגקייטן, און פאדערט דינאמישע פראבלעם-לייזונג.

די שריט צו באהאנדלען קאמפליצירטע פראגעס:

- צעטיילן די פראגע (query decomposition): ערשטנס צעברעכט מען די פראגע אין קלענערע אונטער-פראגעס וואס מען קען באהערשן. למשל, אויב די פראגע נעמט אריין עטליכע פארגלייכן, נעמט די סיסטעם ארויס יעדן פארגלייך אלס באזונדערע ארבעט.
- אויסקלייבן געצייג און פונקציע: יעדע אונטער-פראגע קען פאדערן אנדערע געצייג אדער פונקציעס. צום ביישפיל, ברענגען פינאנציעלע דאטא קען פאדערן א רוף צו א וועב-API, בשעת א פארגלייך קען פאדערן א סטאטיסטישע פונקציע.
- איבערחזרדיגע פראבלעם-לייזונג (iterative): דער אגענט מוז אפשאצן די רעזולטאטן נאך יעדער אונטער-פראגע. אויב דער רעזולטאט הענגט אפ אין עטליכע פאקטארן, קען דער אגענט דארפן צוריקגיין און פארענדערן פריערדיגע שריט.

א ביישפיל: פאָרויסזאגן פארקויף-וואוקס (16:28:35)

די פראגע: "זאג פאראויס דעם וואוקס אין פארקויף פארן קומענדיגן קוואַרטאל, באזירט אויף די פארקויף-דאטא פון די לעצטע דריי קוואַרטאלן, און פארגלייך עס מיט די טרענדס פון דער זעלבער צייט פארגאנגען יאר." די שריט:

- שריט 1: ברענגען די פארקויף-דאטא פון די לעצטע דריי קוואַרטאלן.
- שריט 2: אנאליזירן די דאטא צו געפינען די טרענדס.
- שריט 3: ניצן סטאטיסטישע מאדעלן צו פאָרויסזאגן די צוקונפטיגע פארקויפן.
- שריט 4: ברענגען דאטא פון דער זעלבער צייט פארגאנגען יאר.
- שריט 5: פארגלייכן היי-יעריגע פאָרויסזאגונג מיט פארגאנגען-יעריגע רעזולטאטן.

דער אגענט דארף אינטעליגענט דורכגיין יעדן שריט, זיך צופאסן לויט נייע אינפארמאציע, און אפשר צוריקקערן צו פריערדיגע שריט אויב די דאטא איז נישט גענוג.

א פולער אגענט-טראכט-ציקל אין LlamaIndex (16:29:06)

ביז יעצט, ווערט ערקלערט, זענען די פראגעס געווען געטון אין איין "פארווערטס-גאנג": מען גיט א פראגע, מען רופט דעם ריכטיגן געצייג מיט די ריכטיגע פאראמעטערס, און מען באקומט צוריק דעם ענטפער. אבער וואס

אויב דער באניצער פרעגט א קאמפליצירטע פראגע מיט עטליכע שריט, אדער א פראגע וואס פאדערט א פרטימדיגן ענטפער? אין דער לעקציע לערנט מען צו דעפינירן א פולן אגענט-ציקל פון טראכטן (reasoning loop), אנשטאט רופן א געצייג אין איין שאס. אזא אגענט קען טראכטן איבער געצייג און איבער עטליכע שריט. מען ניצט א "function calling agent", וואס איז אינטעגרירט נאטירלעך מיט די פונקציע-רופן-מעגלעכקייטן פון גרויסע שפראך-מאדעלן (LLM).

די שריט אין קאוד:

- מען שטעלט איין דעם OpenAI API-שליסל.
- מען אימפארטירט nest_asyncio און לויפט עס.
- מען ניצט ווידער דעם MetaGPT פאפיר, ווי אין די צוויי פריערדיגע לעקציעס.
- מען שטעלט איין דעם מאדעל GPT-3.5 Turbo מיט temperature=0.
- מען אימפארטירט FunctionCallingAgentWorker און AgentRunner פון LlamaIndex.
- דעם FunctionCallingAgentWorker גיט מען צוויי געצייג: דעם vector tool און דעם summary tool; מען גיט אויך דעם LLM, און מען שטעלט verbose=True כדי צו זען די צווישן-רעזולטאטן.

די ערשטע פראגע: "דערציילּ מיר וועגן די אגענט-ראלעס אין MetaGPT, און ווי זיי רעדן זיך איבער איינער מיטן אנדערן." אין רעזולטאט זעט מען אז דער אגענט צעטיילט די גאנצע פראגע אין שריט: דער ערשטער טייל פרעגט וועגן די אגענט-ראלעס אין MetaGPT, און ער רופט דעם summary tool צו ענטפערן דערויף.

דערנאך קוקט מען אויף דעם אינהאלט פון דעם ערשטן "source node" וואס איז צוריקגעקומען, מיט metadata mode, כדי צו באקומען די מעטא-דאטא פון יעדן ענטפער. דאס דערמעגלעכט אויספארשן דעם אינהאלט פונעם ערשטן נאוד וואס איז געברענגט געווארן.

שמועס מיט זכרון: agent.chat (16:31:27)

יעצט ניצט מען agent.chat צום ערשטן מאל. די פראגע איז וועגן די עוואלואציע-דאטא-סעטס וואס ווערן גענוצט. עס קען נעמען א ווייל. אין רעזולטאט זעט מען: עס איז צוגעלייגט געווארן צום זכרון די באניצער-מעסעדזש "tell me about the evaluation datasets used", און עס ווערט גערופן דער summary tool פון MetaGPT מיט אן input . דער ענטפער פון דער פונקציע: די עוואלואציע-דאטא-סעטס אין MetaGPT נעמען אריין HumanEval, MBPP און א ווייכווארג-אנטוויקלונג דאטא-סעט. דער LLM-ענטפער איז דער זעלבער, און ער איז ריכטיג.

די קומענדיגע פראגע, ווידער מיט agent.chat אנשטאט query: וועגן די רעזולטאטן אויף די אויבנדערמאנטע דאטא-סעטס. דאס ווייזט די מעגלעכקייט פון היטן די היסטאריע פונעם שמועס, ווייל מען פרעגט א נאכפאלג-פראגע. מען זעט באזונדער: דעם פונקציע-רוף, דעם פונקציע-רעזולטאט, און דעם LLM-ענטפער; דער פונקציע-רעזולטאט און דער LLM-ענטפער דארפן זיין דאס זעלבע.

דער נידעריג-שטאפלדיגער API: שריט פאר שריט (16:33:00)

מען דעפינירט ווידער דעם אגענט דורך FunctionCallingAgentWorker און AgentRunner, דאס מאל גיט מען דעם agent worker אריין אין AgentRunner. יעצט ניצט מען דעם נידעריג-שטאפלדיגן API: מען שאפט א task-אביעקט פון דער באניצער-פראגע, און דערנאך לויפט מען דורך די שריט איינס ביי איינס, מיט דער מעגלעכקייט אפילו אריינצולייגן אייגענעם input.

די פראגע איז ווידער: "דערצייל מיר ווען די אגענט-ראלעס אין MetaGPT און ווי זיי רעדן זיך איבער." מען לויפט נאר איין שריט:

```
agent.run_step(task.task_id)
```

אין רעזולטאט: עס איז צוגעלייגט געווארן צום זכרון, און עס איז דא איין פונקציע-רוף, דער summary tool פון MetaGPT, מיטן input "agent roles in MetaGPT". דער פונקציע-רעזולטאט: די אגענט-ראלעס אין MetaGPT נעמען אריין דעם Product Manager א.א.וו., און דער ענטפער איז ריכטיג. מען זעט אז דער אגענט האט ערלעדיגט נאר דעם ערשטן טייל פון דער פראגע און זיך דארט אפגעשטעלט.

- מיט agent.get_completed_steps(task.task_id) קוקט מען אויף די לאגס און זעט אז דער ערשטער טייל איז דורכגעפירט געווארן, און וויפיל שריט זענען ערלעדיגט.
- מיט agent.get_upcoming_steps(task.task_id) דרוקט מען אויס די צאל קומענדיגע שריט פאר דער ארבעט.

דערנאך לויפט מען די קומענדיגע שריט, מיט דער מעגלעכקייט אריינצוגעבן א באניצער-input. מען זעט אז עס איז צוגעלייגט צום זכרון, און דער קומענדיגער רוף איז מיטן input "how agents share information in MetaGPT". דער פונקציע-רעזולטאט: אגענטן אין MetaGPT טיילן אינפארמאציע דורך א סטרוקטורירטן קאמוניקאציע-פראטאקאל. דאס איז דער לעצטער שריט; וועו מען לויפט אים, באקומט מען גלייך דעם ענטפער. די גאנצע ארבעט איז געטון, און עס בלייבט נאר צוריקצוגעבן איין לעצטן שריט צו צונויפשטעלן דעם ענטפער. צום סוף רופט מען finalize_response מיט דעם task_id, און מען דרוקט אויס דעם ענטפער.

אן אגענט איבער עטליכע דאקומענטן (16:36:04)

די קומענדיגע לעקציע ווייזט ווי צו בויען אן אגענט איבער עטליכע דאקומענטן. מען שטעלט ווידער איין דעם OpenAI API-שליסל און nest_asyncio. די ציל: אויפשטעלן א function calling agent איבער דריי פאפירן, דורך צונויפנעמען די vector- און summary-געצייג פון יעדן דאקומענט אין איין ליסטע, אזוי אז דער אגענט האט אין גאנצן זעקס געצייג.

- מען האט אראפגעלאדן דריי פאפירן: MetaGPT, LongLoRA און Self-RAG.
- א הילפס-פונקציע get_tools בויט אויטאמאטיש פאר יעדן פאפיר א) vector tool וועקטאר-זוך) און א) summary toolסיכום פונעם גאנצן דאקומענט).
- מען לייגט אלע געצייג אין א פלאכע ליסטע.
- מען דעפינירט GPT-3.5 Turbo פון OpenAI אלס LLM.
- מען קאנטראלירט len פון די געצייג: זעקס — דריי פאפירן מאל צוויי געצייג. ריכטיג.

דערנאך בויט מען דעם גאנצן agent worker מיט אלע זעקס געצייג און דעם LLM, און מען קען פרעגן פראגעס איבער אלע דאקומענטן אדער אין איין דאקומענט. די פראגע: "גיב מיר א סיכום פון ביידע, Self-RAG און LongLoRA." אין רעזולטאט: די מעסעדזש איז צוגעלייגט צום זכרון, איין פונקציע-רוף גיט צוריק וועגן Self-RAG און א צווייטער וועגן LongLoRA. מען דערמאנט אויך אז איינער פון די עוואלואציע-דאטא-סעטס איז PG19 test split, און מען קען זען די עוואלואציע-רעזולטאטן פאר די LongLoRA מאדעלן. מען דרוקט אויס דעם ענטפער מיט print((str(response)), און מען באקומט סיכומים פון ביידע פאפירן.

פארבארייטן 11 פאפירן (16:39:11)

צום סוף ווערט געוויזן קאוד וואס דעפינירט צוויי ליסטעס: urls און papers. די urls ליסטע אנטהאלט בערך 11 לינקס צו פארש-פאפירן אין PDF-פארמאט פון דעם OpenReview וועבזייטל, און די papers ליסטע אנטהאלט די פאסיגע טעקע-נעמען, וואס שטעלן מסתמא פאר די אראפגעלאדענע ווערסיעס. יעדער URL פאסט צו א געוויסן טעקע-נאמען, און די נעמען ווערן גענוצט צו אפהיטן די אראפגעלאדענע טעקעס. אזא איינשטעלונג איז נוצלעך צו אויטאמאטיזירן דאס אראפלאדן, היטן חשבון פון די טעקעס, אדער אנווייזן אויף פאפירן לויט זייערע נעמען. מען לויפט דעם קאוד.

מולטי-דאקומענט אגענט און אריינפיר אין RAG (16:40:12)

אויסציען געצייג פאר יעדן פאפיר (16:40:12)

אין דעם טייל פון דער לעקציע ווערט געוויזן ווי אזוי מען פראצעסירט א ליסטע פון פארשונג-פאפירן, און מען ציט ארויס פאר יעדן דאקומענט זיינע אייגענע געצייג דורך דער פונקציע get_doc_tools.

דער קאוד הייבט אן מיט אימפארטירן get_doc_tools פונעם מאדול utils, און Path פון דער ביבליאטעק pathlib, כדי צו באהאנדלען די נעמען פון די טעקעס.

דערנאך מאכט מען א ליידיגן דיקשאנערי וואס הייסט paper_to_tools. אין אים היט מען אפ פאר יעדן פאפיר וועלכע געצייג געהערן צו אים.

דער קאוד גייט דורך די ליסטע papers. ביי יעדן פאפיר דרוקט ער אויס א מעלדונג וועלכער פאפיר ווערט יעצט פראצעסירט.

פאר יעדן פאפיר רופט ער get_doc_tools און גיט איבער צוויי זאכן: דעם נאמען פון דער טעקע, און דעם נאמען אן דער ענדונג ".pdf".

די פונקציע גיט צוריק צוויי געצייג:

- א vector tool. דאס איז א געצייג וואס פארוואנדלט דעם טעקסט אין embeddings פאר NLP-ארבעט.
- א summary tool. דאס איז א געצייג וואס קען מאכן א קורצע צוזאמענפאסונג פון דעם פאפיר.

די געצייג ווערן אריינגעלייגט אינעם דיקשאנערי. דער שליסל איז דער נאמען פון דער טעקע, און דער ווערט איז די געצייג.

אויף דעם אופן קען מען גרינג און שנעל צוקומען צו די וועקטאר-פארמען און צו די צוזאמענפאסונגען פון יעדן פאפיר. מען זעט ווי דער קאוד לויפט און ברענגט ארויס אלע רעזולטאטן.

צונויפזאמלען אלע געצייג אין איין ליסטע (16:41:12)

דער קומענדיגער קאוד שאפט א ליסטע מיטן נאמען all_tools . זי זאמלט צונויף אלע געצייג פונעם דיקשאנערי paper_to_tools, פון יעדן פאפיר אין דער ליסטע.

מען ניצט דערפאר א "nested list comprehension", דאס הייסט א ליסטע-באפעל וואס איז געבויט אינעווייניג אין א צווייטן. ער גייט דורך יעדן פאפיר, נעמט די צוויי געצייג וואס זענען פארהיטן ביי אים אין paper_to_tools, און לייגט יעדן געצייג צו צו all_tools.

דער רעזולטאט איז אז all_tools ווערט איין איינציגע ליסטע. זי אנטהאלט אלע vector און summary געצייג פון אלע פאפירן, און זיי זענען מער נישט צעטיילט לויט פאפיר.

דער קורס ערקלערט אז אזוי איז גרינגער צו ארבעטן מיט אלע געצייג צוזאמען, אנשטאט באהאנדלען זיי פאפיר נאך פאפיר.

בויען א זוכבארן אינדעקס פון די געצייג (16:42:12)

דערנאך בויט מען א אינדעקס און א retriever פאר די געצייג אין all_tools. דער אינדעקס איז געבויט אויף אביעקטן, און מען ניצט דערפאר VectorStoreIndex פון דער ביבליאטעק llama_index.core.

די פונקציע ObjectIndex.from_objects נעמט אלע געצייג און ארדנט זיי אין א זוכבארן אינדעקס.

דער ארגומענט index_cls=VectorStoreIndex זאגט אז דער אינדעקס זאל ניצן בלויז זוכן לויט וועקטארן. דאס הייסט אז יעדער געצייג ווערט פארוואנדלט אין א וועקטאר, כדי מען זאל קענען זוכן שנעל און עפעקטיוו.

אזוי קען מען שנעל טרעפן די פאסיגע געצייג דורך זוכן ענלעכקייט. דאס איז נוצלעך, למשל, צו טרעפן געצייג וואס פאסן צו דער פראגע פונעם באניצער.

דער retriever מיט similarity_top_k (16:42:43)

דער קומענדיגער שורה קאוד איז:

obj_index.as_retriever(similarity_top_k=3) = obj_retriever

דאס שאפט א retriever-אביעקט פונעם אינדעקס. דער אינדעקס איז א צוזאמענשטעל פון דאטא, ווי דאקומענטן אדער embeddings.

דער מעטאד as_retriever פארוואנדלט דעם אינדעקס אין א retriever, וואס ברענגט פאסיגע רעזולטאטן לויט ענלעכקייט.

דער פאראמעטער similarity_top_k=3 זאגט אז ביי יעדער פראגע ברענגט דער retriever די דריי מערסט ענלעכע און פאסיגע רעזולטאטן.

דער קורס ערקלערט אז דאס איז וויכטיג אין סיסטעמען ווי זוך-מאשינען, chatbots אדער רעקאמענדאציע-סיסטעמען. ביי זיי איז זייער וויכטיג צו ברענגען די מערסט פאסיגע אינפארמאציע.

דערנאך ווערט געשריבן tools = (...)obj_retriever.retrieve מיט א פראגע. דער retriever זוכט אין דער אינדעקסירטער דאטא און ברענגט די מערסט ענלעכע רעזולטאטן.

ווייל ער איז אויסגעשטעלט אויף top 3, ברענגט ער ביז דריי שטיקלעך פאסיגע אינפארמאציע. די רעזולטאטן ווערן אריינגעלייגט אינעם וועריעבל tools.

מען קען זיי ניצן ווייטער, למשל צו וועזן דעם באניצער, אדער איבערגעבן צו אן AI מאדעל. דער קורס באמערקט אז אזא פראצעס ווערט אפט גענוצט אין AI זוך-סיסטעמען, chatbots און אפליקאציעס וואס ענטפערן אויף פראגעס.

קוקן אויף די metadata (16:44:16)

דערנאך קוקט מען אויף די metadata פונעם דריטן רעזולטאט. מען שרייבט tools[2].metadata און מען לאזט עס לויפן.

יעדער אביעקט אין tools אנטהאלט סיי דעם אינהאלט וואס איז געבראכט געווארן, סיי צוגעלייגטע metadata.

metadata אנטהאלט געווענליך עקסטערע אינפארמאציע וואס באשרייבט די דאטא, למשל:

- דער מקור,
- די צייט (timestamp),
- דער מחבר,
- דער ID פונעם דאקומענט,
- אנדערע אייגנשאפטן.

בויען דעם אגענט (16:45:19)

יעצט שטעלט מען אויף אן אגענט מיט AI, וואס קען ענטפערן פראגעס לויט אינפארמאציע וואס ער ברענגט פון א צוזאמענשטעל פאפירן.

ערשטנס אימפארטירט מען FunctionCallingAgentWorker און AgentRunner פון llama_index.core.agent.

דערנאך שאפט מען א agent worker דורך FunctionCallingAgentWorker.from_tools. מען גיט אים צוויי זאכן:

- דעם obj_retriever, וואס דינט אלס געצייג צו ברענגען פאסיגע אינפארמאציע,
- דעם llm, וואס פראצעסירט און שרייבט די ענטפערס.

דער system prompt זאגט דעם אגענט אז ער זאל זיך פארלאזן אויף די דאטא וואס ער ברענגט, און נישט אויף זיין פריערדיגע וויסן.

די באשטימונג verbose=True ווייזט א דעטאלירטן לאג. דאס העלפט צו פאררעכטן פעלערן אדער נאכפאלגן ווי אזוי דער אגענט ברענגט און פראצעסירט די דאטא.

צום סוף שרייבט מען agent = AgentRunner(...). דאס וויקלט איין דעם worker אין א AgentRunner, און עס ווערט גרינגער צו פירן און לויפן די אויפגאבעס פונעם אגענט.

דער קורס זאגט אז דער אגענט קען אזוי מיט שכל ברענגען און שאפן ענטפערס פון א פריער-באשטימטער וויסנס-באזע. דאס איז נוצלעך פאר פארשונג, אנאליז פון דאקומענטן, און ענטפערן אויף פראגעס.

פראגעס צום אגענט: MetaGPT, SWE-bench, LongLoRA און LoftQ (16:46:20)

די ערשטע פראגע פרעגט וועלכער evaluation דאטא-סעט ווערט געניצט אין MetaGPT, און ווי אזוי ער פארגלייכט זיך מיט SWE-bench.

די פונקציע query שיקט די פראגע צום אגענט. ער ברענגט פאסיגע אינפארמאציע דורך דעם obj_retriever, און פראצעסירט דעם ענטפער מיטן גרויסן שפראך-מאדעל.

דער ענטפער ווערט אריינגעלייגט אין response. דערנאך פארוואנדלט ((print(str(response דעם ענטפער אין טעקסט און ווייזט אים. אזוי באקומט מען א גוט-אויסגעשטעלטן ענטפער, מיט א פארגלייך פון ביידע דאטא-סעטן וואס איז באזירט אויף די דאקומענטן.

דער לעצטער שריט פרעגט דעם אגענט צו אנאליזירן און פארגלייכן צוויי פארשונג-פאפירן: LongLoRA און LoftQ.

די פראגע איז אויסגעשטעלט אין צוויי טיילן. ערשט זאל דער אגענט אנאליזירן דעם צוגאנג פון יעדן פאפיר. דערנאך זאל ער זיי פארגלייכן און ווייזן דעם אונטערשייד.

דער אגענט ברענגט די אינפארמאציע דורך דעם retriever, פראצעסירט זי מיטן שפראך-מאדעל, און דער ענטפער ווערט אריינגעלייגט אין response. מען קען אים אויסדרוקן אדער ניצן ווייטער.

עס ווערט ערקלערט אז דאס העלפט ארויסצוציען פארשטאנד פון פארשונג-פאפירן אויף אן עפעקטיוון אופן. דאס איז נוצלעך פאר אקאדעמישע פארשונג, טעכנישע אנאליז, און צוזאמענפאסונגען מיט AI.

סיכום פונעם agentic RAG קורס (16:47:52)

דער קורס פאסט צוזאמען דעם גאנצן וועג. מען האט געלערנט ווי אזוי צו פארבעסערן דעם AI'ס פעאיגקייט צו ברענגען, פראצעסירן און דורכטראכטן קאמפליצירטע אינפארמאציע אויף א דינאמישן אופן.

די שריט אויפן וועג זענען געווען:

- אן אריינפיר: פארוואס agentic reasoning איז וויכטיג אין retrieval augmented generation, און ווי אזוי עס העלפט באהאנדלען פארשידענע סארטן פראגעס.
- דער router query engine: ער שיקט יעדע פראגע צום ריכטיגן מקור, און מאכט אזוי דאס ברענגען פון אינפארמאציע מער עפעקטיוו.
- tool calling: AI אגענטן קענען ארבעטן מיט דרויסנדיגע געצייג, און באשליסן מער ווי בלויז שרייבן טעקסט.
- בויען אן agent reasoning cycle: א סיסטעם וואס פראצעסירט אינפארמאציע אין רונדן, און פארבעסערט זיינע ענטפערס לויט נייע אנזעענישן.
- א multi-document agent: ער ציט צוזאם און פאראייניגט אינפארמאציע פון עטליכע מקורות, און גיט אזוי מער פולשטענדיגע און גענויע רעזולטאטן.

אלע די טיילן קומען צוזאמען אין אן אינטעליגענטן agentic RAG סיסטעם. אזא סיסטעם קען באשליסן, ברענגען פאסיגע אינפארמאציע, און ארבעטן מיט געצייג.

דער קורס דערמאנט אז די מעגלעכקייטן זענען ברייט: פערזענליכע AI געהילפן, אויטאמאטישע פארשונג-געצייג, און סיסטעמען וואס באשליסן לויטן קאנטעקסט.

ווען AI אנטוויקלט זיך ווייטער, זאגט דער קורס, ווערט דאס פאראייניגן פון דענקען, ברענגען אינפארמאציע און ארבעטן מיט געצייג זיין א שליסל צו מער פעאיגע, זעלבשטענדיגע און צופאסבארע סיסטעמען.

מען מוטיגט צו עקספערימענטירן מיט agentic RAG, צו פארבעסערן ווי דער אגענט באשליסט, און צו זוכן נייע וועגן פאר אויטאמאציע און בעסערן פארשטאנד פונעם קאנטעקסט.

וואס איז RAG? (16:49:55)

דא הייבט זיך אן א נייער טייל. RAG מיינט retrieval augmented generation . דאס איז א וועג צו מאכן אן AI מאדעל ענטפערן מיט דרויסנדיגע אינפארמאציע, אנשטאט זיך פארלאזן בלויז אויף דעם וואס ער ווייסט שוין.

RAG ארבעט אין צוויי שריט:

- ערשטנס זוכט ער פאסיגע אינפארמאציע. ער זוכט אין PDF טעקעס, דאטא-באזעס, וועבזייטלעך, וויסנס-באזעס, אדער טעקעס פון דער פירמע.
- דערנאך ניצט ער די געבראכטע אינפארמאציע צו שרייבן דעם ענדגילטיגן ענטפער.

מיט אנדערע ווערטער: אנשטאט "ענטפער בלויז פון דיין טריינירן", זאגט RAG: "קוק קודם נאך די ריכטיגע אינפארמאציע, און ענטפער דערפון".

פארוואס איז RAG נוצלעך? א געוועלטיגער מאדעל קען פארגעסן פרטים, גלויבן אן אלטמאדישן ענטפער, און ראטן וועלן ער איז נישט זיכער.

RAG העלפט דערמיט. ער קען ניצן די לעצטע אינפארמאציע און די דאקומענטן פון דער פירמע. ער פארמינערט hallucination, דאס הייסט וועלן דער מאדעל דערפינדט זאכן. און ער גיט ענטפערס פון א פארלעסלעכן מקור.

ביישפיל: HR פאליסי פון א פירמע (16:51:28)

לאמיר נעמען א ביישפיל. מען פרעגט: "וויפיל casual leaves (פריי-טעג) באקומען ארבעטער א יאר?"

א געוועלטיגער מאדעל, ווי ChatGPT, קען גלויבן אן אלגעמיינעם HR ענטפער.

מיט RAG קוקט דער סיסטעם קודם אין דעם HR פאליסי-דאקומענט פון דער פירמע. ער טרעפט, למשל, אז ארבעטער אויף פולע צייט באקומען 12 casual leaves א יאר. דערנאך ענטפערט ער: "ארבעטער באקומען 12 casual leaves א יאר". דער ענטפער איז באזירט אויפן דאקומענט, און נישט אויף א טרעף.

א ביישפיל פון דער וועלט: א פירמע "XYZ" בויט אן אינערליכן HR chatbot. ארבעטער פרעגן פראגעס ווי:

- וויפיל קראנק-טעג באקום איך?
- קען איך איבערטראגן casual leave אויפן קומענדיגן יאר?
- ווי אזוי בעט מען maternity leave?
- וואס איז די notice period?

אן RAG קען דער AI גלויבן זייער אלגעמיינע ענטפערס ווי פונעם אינטערנעט. ער קען צעמישן די פאליסי פון איין פירמע מיט אן אנדערער, אדער גלויבן א פאלשן ענטפער.

מיט RAG זוכט דער סיסטעם אין דעם HR פאליסי PDF, טרעפט דעם גענויעם אפטייל וואס שייך צו דער פראגע, און ענטפערט לויט יענעם אפטייל.

הינטער די קוליסן ווערן די HR פאליסיס אפגעהיטן אינעם סיסטעם. פרעגט דער באניצער "קען מען איבערטראגן casual leave?", ברענגט RAG דעם שורה: "casual leaves קענען נישט איבערגעטראגן ווערן". דערפאר, זאגט דער קורס, איז RAG אזוי וויכטיג אין מאדערנע AI אפליקאציעס.

דער RAG פלוס (16:53:33)

כדי צו פארשטיין די ארכיטעקטור, ווערט געוויזן דער גאנג פון א פראגע ביז צום ענטפער:

- דער באניצער פרעגט: "קען מען איבערטראגן א casual leave?"
- שריט 1, retrieval) דער זוך-שריט): דער סיסטעם זוכט אין די HR פאליסי-דאקומענטן נאך פאסיגער אינפארמאציע.
- ער ברענגט די פאסיגע פאליסי: "casual leaves קענען נישט איבערגעטראגן ווערן".
- שריט 2, generation: דער AI ניצט וואס ער האט געבראכט, כדי צו שאפן דעם ענדגילטיגן ענטפער. דאס פאסירט אינעווייניג.
- דער ענטפער: "ניין, casual leaves קענען נישט איבערגעטראגן ווערן".

די סטרוקטור פונעם פראיעקט (16:54:36)

צום סוף הייבט מען אן ווייזן ווי די סטרוקטור פונעם פראיעקט וועט אויסזען:

- אין דעם הויפט-פאלדער איז דא א פאלדער data.
- אין data איז דא אן אונטער-פאלדער raw. דארט ליגט דער פולער וויסנס-מקור, א טעקע company_policy. אין איר שטייט די דרויסנדיגע אינפארמאציע וועגן די פאליסיס פון דער פירמע.
- דערנאך שאפט מען נאך א פאלדער מיטן נאמען processed.

RAG-פראיעקט סטרוקטור, פיין-טיונינג און פראמפט-אינזשעניערינג (16:55:08)

די פאלדער-סטרוקטור פון א RAG אפליקאציע (16:55:08)

דער קורס בויט ווייטער אויף די פאלדער-סטרוקטור פון א RAG פראיעקט. מען שאפט א טעקע וואס הייסט chunk.json , וואס געהערט צו די פראצעסירטע דאטא. דערנאך מאכט מען א סוב-פאלדער וואס הייסט vector store, און אין אים נאך א טעקע אדער פאלדער פאר) FAISSאין די קאפשאנס אויסגעהערט ווי "fis").

דערנאך שאפט מען א פאלדער וואס הייסט src, וואס איז דער הויפט-פאלדער פון גאנצן פראיעקט. אין src מאכט מען די פאלגנדע טעקעס:

- loader.py
- chunker.py
- embedder.py
- retriever.py
- generator.py
-) rag_pipeline.pyאין אנהייב איז די טעקע פארגעסן געווארן, און שפעטער האט מען זי צוגעלייגט)

• utils.py

נאכדעם שאפט מען א פאלדער וואס הייסט prompts, און אין אים א טעקע rag_prompt.txt. אין הויפט-פאלדער לייגט מען צו (app.py, config.py, requirements.txtכדי צו וויסן וועלכע דעפענדענסיס דער פראיעקט דארף), אן environment variable טעקע (.env) צו האלטן אלע סודות ווי שליסלען, און א טעקע test_query.py. דאס איז די גאנצע פאלדער-סטרוקטור פון א RAG אפליקאציע.

וואס יעדער פאלדער און טעקע טוט (16:57:12)

דער קורס ערקלערט יעדן טייל באזונדער:

• דער data פאלדער: דא האלט מען די אריגינעלע דאקומענטן, למשל PDF טעקעס, טעקסט-טעקעס אדער דאקומענטאציע.

• דער processed פאלדער: דא ליגן די ריינע טעקסט-שטיקער (chunks) אדער דער ארויסגעצויגענער אינהאלט. למשל פון א HR policy PDF, א company handbook, א product manual אדער אן FAQ דאקומענט.

• דער vector store אדער FAISS פאלדער: דא האלט מען די embeddings און די אינדעקס-טעקעס, למשל א FAISS index, ChromaDB, Pinecone metadata אדער Weaviate storage. עס קלינגט א ביסל טעכניש, אבער דאס איז דער פלאץ וואו די סיסטעם זוכט די שטיקער וואס פאסן צו דער פראגע.

• דער src פאלדער: דא ליגט די הויפט-לאגיק פון RAG.

• loader.py לאדנט די דאקומענטן פון די טעקעס. ער לייענט PDF, ציט ארויס דעם טעקסט, און לאדנט טעקעס פון סארט txt אדער docx.

• chunker.py צעטיילט לאנגע טעקסטן אויף קלענערע שטיקער. למשל, ער שניידט יעדע 500 ווערטער, מיט אן איבערלאפ (overlap) פון 50 ווערטער.

• embedder.py פארוואנדלט די טעקסט-שטיקער אין embeddings, למשל מיט OpenAI embeddings, Hugging Face embeddings אדער Sentence Transformers.

• retriever.py געפינט די שטיקער וואס פאסן אמבעסטן צו דער פראגע פונעם באניצער. למשל, א similarity search וואס ברענגט די דריי שטיקער וואס פאסן אמבעסטן (top 3).

• generator.py שיקט צום LLM דעם צוריקגעברענגטן קאנטעקסט צוזאמען מיט דער פראגע פונעם באניצער. ער בויט דעם לעצטן פראמפט און באקומט דעם ענטפער פונעם מאדעל.

• rag_pipeline.py פארבינדט אלעס צוזאמען. דער פלוס גייט אזוי: ערשט לאדנט מען די פראגע, דערנאך ברענגט מען די שטיקער, און צום סוף גענערירט מען דעם ענטפער.

• utils.py האלט אלע הילפס-פונקציעס וואס דער פראיעקט דארף, למשל ריינמאכן טעקסט, logging, אדער פארמאטירן דעם אויסגאנג.

• דער prompts פאלדער איז איינער פון די וויכטיגסטע. ער האלט די פראמפט-מוסטערן (templates). א ביישפיל: "You are a helpful assistant. Answer only from the provided context. If the answer is not found, say: not found in the document."

• app.py איז דער הויפט-אריינגאנג פון דער אפליקאציע. דאס קען זיין א CLI אפליקאציע, א Streamlit אפליקאציע, א Gradio אפליקאציע, אדער א FastAPI backend.

- config.py האלט די קאנפיגוראציע-ווערטן, למשל דעם נאמען פונעם מאדעל, די chunk size , דעם overlap, און דעם וועג (path) צום vector DB.
- requirements.txt האלט אלע Python ביבליאטעקן וואס מען דארף צו בויען די RAG אפליקאציע.
- test_query.py איז א קליינער סקריפט צו טעסטן די RAG סיסטעם מיט א ביישפיל-פראגע.

דער פשוטער פלוס פון דער סטרוקטור זעט אויס אזוי: דער דאקומענט גייט צום loader , פון דארט צום chunker, וואס צעטיילט דעם טעקסט אויף שטיקער, דערנאך צום embedder, און צום סוף צום vector store. אט אזוי זעט אויס א RAG פראיעקט.

וואס איז פיין-טיונינג (17:01:50)

דערנאך גייט דער קורס איבער צו פיין-טיונינג (fine-tuning). מען עפנט די אפיציעלע Google developers guide, וואו עס שטייט אז פיין-טיונינג איז א וועג צו מאכן א פריער-טריינירטן מאדעל בעסער אין א ספעציפישער אויפגאבע, פארמאט אדער סטיל, דורכן אים ווייטער טריינירן. דערבייַ קען מען צושטעלן אייגענע ביישפילן.

OpenAI באשרייבט עס אזוי: מען גיט דעם באזע-מאדעל די סארט אינפוטס און אויטפוטס וואס מען ערווארט אין דער אפליקאציע, און מען באקומט א מאדעל וואס ארבעט בעסער אויף יענע אויפגאבעס. Google'ס Vertex AI דאקומענטאציע זאגט כמעט דאס זעלבע. זי באשרייבט supervised fine-tuning ווי צופאסן דאס אויפפירונג פונעם מאדעל מיט א באצייכנטן דאטא-סעט (labeled data), דורכן ענדערן די) twightsדי ווייטס) פונעם מאדעל.

פשוט ערקלערט: א גרויסער שפראך-מאדעל קען שוין די אלגעמיינע שפראך-מוסטערן, ווייל מען האט אים פריער טריינירט אויף א סך דאטא. פיין-טיונינג גיט אים נאך א רונדע לערנען אויף א קלענערן און געצילטן דאטא-סעט, אזוי אז ער ווערט בעסער אין עפעס ספעציפיש. למשל: ענטפערן אין דעם טאן פון דער פירמע, האלטן זיך צו א פעסטן אויסגאנג-פארמאט, אדער טאן א געוויסע קלאסיפיקאציע אדער סומירונג. אזוי ווערט ער מער קאנסעקווענט ביי א שמאלן ביזנעס-פראצעס.

דער משל: דער באזע-מאדעל איז ווי א קלוגער גראדואנט, און פיין-טיונינג איז די טריינינג פאר א ספעציפישער ארבעט. דער מאדעל לערנט זיך נישט פון אנהייב. מען מאכט אים נאר ספעציאליזירט.

וואס ענדערט זיך פאקטיש? די אינערליכע פאראמעטערס, אדער דאס געלערנטע אויפפירונג פונעם מאדעל, ווערן צוגעפאסט לויט די טריינינג-ביישפילן. היינטיגע פלאטפארמעס שטיצן אויך גרינגערע מעטאדן, ווי supervised fine-tuning און parameter-efficient מעטאדן ווי LoRA אדער QLoRA. די מעטאדן זענען געמאכט צו צופאסן מאדעלן ביליגער ווי א פולע פיין-טיונינג.

א ביישפיל: קאסטומער-סופּפּארט אסיסטענט (17:03:54)

א פירמע וויל אן AI אסיסטענט פאר קאסטומער-סופּפּארט. אן פיין-טיונינג קען דער מאדעל אמאל ענטפערן ריכטיג, אבער דער סטיל טוישט זיך. איין ענטפער קען זיין לאנג, אן אנדערער קען זיין צו לויז, און א דריטער קען אויסלאזן דעם פארמאט פון דער פירמע.

מיט פיין-טיונינג גיט מען אים ביישפילן. למשל, דער אינפוט איז א קאסטומער וואס פרעגט וועגן דעם סטאטוס פון א refund. דער געווינטשענער אויטפוט דארף זיין העפלעך, קורץ, און דער עיקר צוגעפאסט צום בראנד פון דער פירמע. ער דארף זיך אויך האלטן צום סטאנדארט-פארמאט פון דער פירמע. ביים פיין-טיונינג דארף

מען נעמען אין באטראכט די אלע פאראמעטערס. נאך אזעלכע ביישפילן ווערט דער מאדעל פיל מער קאנסעקוועוענט אין ארויסגעבן אזא סארט ענטפער. דאס שטימט מיט דעם געווענליכן באנוץ פון פיין-טיונינג: א ספעציאליזירטע אויפגאבע, דער אויסגאנג-סטיל פונעם LLM, און אויפפירונג לויט די באדערפענישן פון דער ארגאניזאציע.

RAG קעגן פיין-טיונינג (17:05:28)

דער קורס פארגלייכט ווי RAG איז אנדערש פון פיין-טיונינג. RAG גיט דעם מאדעל די ריכטיגע אינפארמאציע אין דער צייט פון לויפן (runtime), דורכן ברענגען זי פון די דאקומענטן. פיין-טיונינג, אנדערש, ענדערט ווי דער מאדעל פירט זיך, ווייל מען האט אים נאכגעטריינירט אויף ביישפילן.

דער כלל: נוצט RAG ווען איר דארפט די לעצטע אינפארמאציע אדער וויסן פון דאקומענטן. נוצט פיין-טיונינג ווען איר דארפט א קאנסעקווענטן סטיל, פארמאט, אדער בעסערע אויספירונג פון אן אויפגאבע.

א ביישפיל: אן HR באט מוז שטענדיג ענטפערן אין א געוויסן פארמאט מיט דריי טיילן: דער ענטפער פון דער פאליסי, וועלכע שריט דער ארבעטער דארף טאן, און א נאטיץ מיט ווי צו פארבינדן זיך מיט HR. אויב מען פיין-טיונט דעם מאדעל אויף א סך HR ביישפילן אין דער פונקטליכער סטרוקטור, קען ער זיך אויסלערנען צו ענטפערן אזוי אלעמאל. אבער אויב די פאליסי טוישט זיך אפט, דארף מען נאך אלץ RAG אדער אן אנדער retrieval סיסטעם פאר דעם לעצטן טעקסט פון דער פאליסי. דאס איז ווייל פיין-טיונינג איז בעסער פאר אויפפירונג און פארמאטירונג, בשעת retrieval איז בעסער פאר וויסן וואס טוישט זיך.

אין איין שורה: פיין-טיונינג מיינט אויסלערנען א פריער-טריינירטן מאדעל זיך צו פירן בעסער פאר אייער ספעציפישן באדארף, דורכן אים טריינירן אויף אייערע אייגענע ביישפילן. מיט RAG, אנדערש, גיט מען דעם מאדעל דעם לעצטן קאנטעקסט, דורכן נאכקוקן אין וויסן-קוואלן, דרויסנדיגע דאטא-באזעס אדער APIs.

וואס איז פראמפט-אינזשעניערינג (17:07:02)

דערנאך גייט מען איבער צו פראמפט-אינזשעניערינג. מען עפנט די אפיציעלע דאקומענטאציע פון OpenAI, און דער קורס רעקאמענדירט צו דורכגיין זי. לויט OpenAI איז פראמפט-אינזשעניערינג דער פראצעס פון פלאנירן און פארבעסערן דעם אינפוט וואס מען גיט אן AI מאדעל, כדי ער זאל ארויסגעבן מער ניצלעכע, פונקטליכע און גוט-סטרוקטורירטע רעזולטאטן. OpenAI באשרייבט עס ווי נוצן סטראטעגיעס צו פארבעסערן די רעזולטאטן פון פראמפטן. Google באשרייבט prompt design ווי שאפן פראמפטן וואס רופן ארויס דעם געווינטשענעם ענטפער פון גרויסע שפראך-מאדעלן.

פשוט געזאגט מיינט עס: ווי אזוי מען פרעגט ביי AI. דער מאדעל קען זיין שטארק, אבער די קוואליטעט פונעם ענטפער הענגט אפט אפ אין עטלעכע זאכן: ווי קלאר מען באשרייבט די אויפגאבע, וויפיל קאנטעקסט מען גיט, צי מען שטעלט אויס דעם פארמאט, און צי מען לייגט אריין ביישפילן אדער פראמעטערס. די אלע זענען געווענליכע כללים און best practices פון פראמפט-דיזיין.

מען קען זיך וועגדן צו OpenAI'ס prompt engineering guide אדער צו Google'ס prompt design guide. ביידע האבן ארויסגעגעבן אפיציעלע דאקומענטאציע. ביי Google זוכט מען "prompt design strategies" און מען קומט צו דער Gemini API דאקומענטאציע. ענליך קען מען זוכן OpenAI prompt design strategies, וואו מען זעט די זעלבע זאכן. אין דער דאקומענטאציע זעט מען יעדן סארט פראמפט מיט דעם אויסגאנג וואס ווערט פון אים געענערירט. מען זעט אויך partial input completion און נאך א סך זאכן. דער קורס רעקאמענדירט דאס דורכצוגיין כדי צו באקומען א בעסערן באגריף.

צום סוף הייבט מען אן ווייזן א פשוטן ביישפיל: מען גייט צו ChatGPT און גיט אים א צופעליגע פראגע, למשל "explain this in simple English", און מען הייבט אן אריינקאפירן א טעקסט.

פראמפט-אינזשעניערינג און מולטימאדאל RAG (17:10:09)

ווי אזוי מען פארבעסערט א פראמפט (17:10:09)

דער קורס הייבט אן מיט א פראקטישן ביישפיל פון טעכנישע זאכן: מען נעמט די דאקומענטאציע פון דעם "low rank fine-tuning" אלגאריטם (LoRA), און מען בעט ChatGPT עס צו ערקלערן. מען קאפירט דעם גאנצן טעקסט, מען לאדט עס ארויף, און מען שרייבט א פשוטן פראמפט. עס ווערט ערקלערט אז אזא פראמפט איז זייער אומקלאר (vague), און עס ווערט געוויזן ווי אזוי מען קען אים פארבעסערן.

דער פארבעסערטער פראמפט זאגט בערך אזוי: "דו ביסט אן AI עקספערט, און איך באציך מיך צו IBM'ס אפיציעלע דאקומענטאציע פאר LoRA, וואס איז א טעכניק פאר fine-tuning פון LLM. ניץ די IBM אינפארמאציע און ערקלער מיט א ביישפיל פונעם עכטן לעבן."

מען זעט אז דער פראמפט איז געווארן אסאך בעסער, ווייל יעצט גיט מען דעם LLM קאנטעקסט, און דערפאר וועט ער צוריקגעבן א פאסיגן ענטפער. עס ווערט באמערקט אז עס זענען דא אסאך סטראטעגיעס ווי אזוי צו פארבעסערן א פראמפט, און דער קורס רעקאמענדירט צוויי מקורות צו שטודירן דערויף (די אנווייזונגען פון OpenAI און Google, וואס ווערן באלד דערמאנט).

פארוואס פראמפט-אינזשעניערינג איז וויכטיג (17:11:13)

דאס גאנצע ווערט אנגערופן "prompt engineering". מען טריינירט נישט דעם מאדעל, נאר מען פירט אים אויף א ריכטיגן אופן.

דער קורס ערקלערט אז א גוט-אויסגעשטעלטער פראמפט קען פארבעסערן:

- די קוואליטעט פונעם ענטפער;
- די פינקטלעכקייט;
- די פולשטענדיגקייט;
- די פארמאטירונג.

פראמפט-אינזשעניערינג איז געווענליך אן איטעראטיווער פראצעס: מען פרובירט א פראמפט, מען קוקט וואס קומט ארויס, און מען פארבעסערט אים לויט די רעזולטאטן.

וואס געהערט אריין אין א גוטן פראמפט (17:11:44)

לויט די אנווייזונגען פאר פראמפט-דיזיין פון OpenAI און Google, אנטהאלטן עפעקטיווע פראמפטן קלארע אינסטרוקציעס, פאסיגן קאנטעקסט, ביישפילן וועו עס פאסט, און דעם געוואונטשענעם פארמאט פון ענטפער.

אין דער לעקציע ווערט געוויזן א סטרוקטור פון די טיילן פון א פראמפט:

- ראלע (role): וועו דער מאדעל זאל זיין, ווי למשל "AI עקספערט";
- אויפגאבע (task): וואס מען וויל אז ער זאל טון;
- קאנטעקסט: די הינטערגרונט-אינפארמאציע;

- באגרעניצונגען (constraints): וועלכע גרעניצן און תנאים זענען דא;
- און פון די אלע טיילן בויט מען דעם פערטיגן פראמפט.

פראמפט-אינזשעניערינג קעגן RAG און fine-tuning (17:12:15)

דער קורס פארגלייכט די דריי מעטאדן אויף א פשוטן אופן. פראמפט-אינזשעניערינג זענען אינסטרוקציעס כדי צו באקומען דעם ענטפער וואס מען דארף. fine-tuning פארבעסערט דאס וואס דער מאדעל האט שוין געלערנט, דורך נאך טריינירן. פראמפט-אינזשעניערינג טוישט די בקשה, נישט דעם מאדעל אליין. RAG און fine-tuning לייזן אנדערע פראבלעמען.

קורץ געזאגט: פראמפט-אינזשעניערינג איז די פראקטיק פון שרייבן און פארבעסערן פראמפטן מיט אכטונג, כדי אז דער AI מאדעל זאל געבן דעם סארט ענטפער וואס מען וויל.

איינפירונג צום מאדול פון מולטימאדאל RAG (17:12:46)

דא הייבט זיך אן א נייער מאדול וועגן מולטימאדאל RAG. דער פלאן איז:

- צו ערקלערן וואס מולטימאדאל RAG איז, זיינע הויפט-באגריפן און וואו מען ניצט עס אין דער פראקטיק;
- דורכצוגיין די ארכיטעקטור: די וויכטיגע טיילן און ווי זיי ארבעטן צוזאמען;
- ווי אזוי צו צוגרייטן און פארארבעטן דאטא, כדי אז ווידעאס, בילדער און טעקסט זאלן זיין גרייט צום אויסזוכן;
- פראקטישע ארבעט מיט א וועקטאר-דאטאבאזע, און ווי אזוי מולטימאדאל embedding retrieval ארבעט;
- מער אנטוויקלטע טעכניקן צו געפינען און צוזאמענשטעלן אינפארמאציע פון בילדער און ווידעאס.

דער ציל איז אז אינעם סוף זאל מען פארשטיין מולטימאדאל RAG פראקטיש, און קענען בויען AI לייזונגען וואס פארבינדן פארשידענע סארטן דאטא.

וואס איז RAG און וואס איז מולטימאדאל RAG (17:13:47)

RAG (Retrieval Augmented Generation) איז אן AI פריימווערק וואס פארבינדט א מעכאניזם פון אויסזוכן (retrieval) מיט א גענעראטיוון מאדעל, כדי צו פראדוצירן מער פינקטלעכע ענטפערס וואס פאסן צום קאנטעקסט. אין א געווענלעכן RAG סיסטעם זוכט א retrieval טייל פאסיגע אינפארמאציע אין א גרויסע זאמלונג פון דאקומענטן אדער דאטא, און דער גענעראטיווער מאדעל ניצט דאס כדי צו שרייבן ענטפערס אדער סיכומים. אזוי ווערט דער ענטפער מער אינפארמירט און פאסיג צום קאנטעקסט.

מולטימאדאל RAG ברייטערט אויס דעם געדאנק דורך אריינברענגען עטליכע סארטן דאטא (modalities), ווי טעקסט, בילדער און טאבעלעס. ווייל דער סיסטעם קען פארארבעטן און פארבינדן אינפארמאציע פון פארשידענע מקורות, איז ער זייער פלעקסיבל און מען קען אים ניצן פאר פילע צוועקן. ער קען פארשטיין און שאפן ענטפערס מיט דער גאנצער אינפארמאציע וואס איז פאראן, און דאס ברענגט מער פולשטענדיגע און פינקטלעכע רעזולטאטן.

די טיילן פון א מולטימאדאל RAG סיסטעם (17:14:47)

דער קורס רעכנט אויס פיר הויפט-טיילן:

• אריינברענגען און פאראַרבעטן דאטא (data ingestion and processing): דער סיסטעם מוז קענען ארבעטן מיט פארשידענע סארטן דאטא, און ארויסנעמען און צוגרייטן אינפארמאציע פון טעקסט, טאבעלעס און בילדער.

• אויסזוך-מעכאניזם (retrieval): דער טייל זוכט פאסיגע דאטא אין א זאמלונג אדער דאטאבאזע, מיט embeddings און אנדערע זוך-אלגאריטמען, כדי צו געפינען די מערסט פאסיגע אינפארמאציע.

• געניראטיוואר מאדעל: א גרויסער שפראך-מאדעל (LLM) ווי GPT-4o אדער Gemini Vision Pro שאפט סיכומים, ענטפערס אדער אנדערע רעזולטאטן לויט דער געפונענער אינפארמאציע.

• פארבינדונג און רעזולטאט: די געפונענע דאטא און דאס וואס דער מאדעל האט געשאפן ווערן צוזאמענגעשטעלט אין איין קלארן ענטפער וואס פאסט צום קאנטעקסט.

א פראקטישע אויספירונג: PDF מיט טעקסט, טאבעלעס און בילדער (17:15:49)

עס וועלן באשריבן אן אויספירונג פון א מולטימאדאל RAG סיסטעם וואס פאראַרבעט און פאסט צוזאם דאטא פון א PDF טעקע: טעקסט, בילדער, טאבעלעס און נאך. די הויפט-קלאס איז געבויט צו ארבעטן מיט עטליכע סארטן דאטא. זי ארבעט מיט א LLM פאר בילדער (vision) און טעקסט, און האט פארשידענע הילפס-פונקציעס פאר סיכומען און פאראַרבעטן. די שריט זענען:

• ארויסנעמען פון PDF: דער מאדול פאר טעקעס צעטיילט דעם PDF דאקומענט אויף זיינע טיילן (טעקסט, טאבעלעס און בילדער), כדי מען זאל זיי קענען וויטער אנאליזירן. ער נעמט דעם וועג (path) צו דער PDF טעקע, און צעטיילט דעם דאקומענט אויף שטיקלעך (chunking) לויט די קעפלעך אדער טיטלען.

• סיכומען: יעדער סארט דאטא האט זיין אייגענעם פראצעס. ביים טעקסט וועלן דער טעקסט אליין צוריקגעגעבן אלס סיכום, ווי א דירעקטער דורכגאנג (pass-through). די טאבעלעס וועלן געסיכומט מיט א פראמפט-מוסטער (template) וואס מען שיקט צו GPT-4o. דאס וועלן געטון אין גרופעס (batch), מיט א באגרעניצונג וויפיל ארבעטן גלייכצייטיג (concurrency limit), כדי צו קענען באהאנדלען אסאך טאבעלעס.

• אפהיטן דאטא אין Milvus: מען גרייט צו די געסיכומטע טיילן אין א פאסיגן פארמאט. יעדער טייל וועלן איינגעפאקט אין א document אביעקט מיט פאסיגע metadata.

• אפהיטן טאבעלעס אין PostgreSQL: מען פארוואנדלט די טאבעלע אין א data frame, מען שטעלט צו די דאטא-טיפן צו טיפן וואס SQL פארשטייט, און מען שאפט א SQL פראגע (query) צו שאפן די טאבעלעס אין דער דאטאבאזע. דערנאך שטעלט מען אריין די דאטא שורה נאך שורה, און מען מאכט commit צו דער טראנזאקציע כדי די דאטא זאל בלייבן. ביים סינכראניזירן שיקט מען די דאטא צו דער דאטאבאזע, מען נעמט זי צוריק, און מען ווייזט זי אלס data frame.

• שאפן ענטפערס: וועלן א פראגע קומט, ניצט דער סיסטעם אן אנדערן ארבעטס-גאנג, לויט צי די פראגע איז וועגן טעקסט, טאבעלעס אדער בילדער. ביי א פראגע וועגן טעקסט זוכט מען סיכומים וואס שטימען מיטן באדייט (semantically), וואס זענען אפגעהיט אלס document אין Milvus. מען ברענגט די דאקומענטן לויט ווי ענלעך זיי זענען צו דער פראגע, און מען ווייזט די רעזולטאטן מיט Streamlit'ס פונקציע צו ווייזן טעקסט.

דער סך-הכל: אזא סיסטעם קען אויף אן עפעקטיוון אופן ענטפערן אויף פארשידענע פראגעס וועגן טעקסט, טאבעלעס און בילדער פון דעם PDF, און די אנטוויקלטע זוך-מעגלעכקייטן זארגן אז די ענטפערס זאלן זיין פינקטלעך און פאסיג צום קאנטעקסט.

די ארכיטעקטור פון מולטימאדאל RAG (17:19:02)

עס ווערט געוויזן א דיאגראם פון א דאטא-ליניע (pipeline) פאר AI וואס זוכט וויסן. מען זעט ווי סטרוקטורירטע און נישט-סטרוקטורירטע דאטא ווערט פארארבעט, פארוואנדלט אין embeddings, און אפגעהיט אין א וועקטאר-דאטאבאזע, כדי מען זאל זי שנעל קענען געפינען.

ערשטנס, די דאטא. די ליניע הייבט זיך אן מיט רויע מקורות:

- טעקסט: דאקומענטן, ארטיקלען, באריכטן אדער סיי וואספארא געשריבענע זאך;
- טאבעלעס: סטרוקטורירטע דאטא, ווי spreadsheets אדער דאטאבאזעס;
- בילדער: וויזועלע דאטא, ווי געסקענטע דאקומענטן, פיקטשערס אדער דיאגראמען.

די מקורות ווערן צעטיילט אויף שטיקלעך אדער געסיכומט, כדי עס זאל זיין גרינגער זיי צו פארארבעטן און אין זיי צו זוכן.

צווייטנס, פארארבעטן און embedding. דא זענען דא צוויי וועגן:

- געווענלעכער embedding: דער ארויסגענומענער טעקסט, די טאבעלעס און בילדער ווערן פארוואנדלט אין געסיכומטע אדער צעטיילטע ווערסיעס פאר embeddings. די רויע ווערסיע פון טאבעלעס און בילדער ווערט אויך אפגעהיט פאר רעפערענץ, און די embeddings ווערן אפגעהיט אין דעם vector store.
- מולטימאדאל embedding: דא באהאנדלט מען נישט טעקסט, טאבעלעס און בילדער באזונדער, נאר מען פארארבעט די אלע פארמאטן גלייכצייטיג. מען שאפט א מולטימאדאל embedding, וואס לאזט דעם AI מאדעל געפינען אינפארמאציע מיט רייכן קאנטעקסט, וואס פארבינדט טעקסט, טאבעלעס און בילדער. די embeddings ווערן אויך אפגעהיט אין דעם vector store.

די מעלות און ווי מען ניצט עס (17:20:34)

דער קורס רעכנט אויס די הויפט-מעלות:

- זוכן לויט באדייט (semantic search): ווייל די דאטא איז פארוואנדלט אין וועקטארן, קענען באניצער געפינען פאסיגע אינפארמאציע לויט דעם באדייט, אן צו דארפן טרעפן די פינקטלעכע ווערטער. למשל, אנשטאט צו זוכן "revenue report Q3", קען מען פרעגן "show me financial trends in Q3", און מען באקומט די פאסיגע דאטא.
- פארבינדן פארשידענע סארטן דאטא: דער AI מאדעל קען פארשטיין טעקסט, טאבעלעס און בילדער צוזאמען, און זען ווי זיי זענען פארבונדן.
- גוט פאר גרויסע אפליקאציעס: עס לאזט שנעל זוכן און געפינען, וואס איז נוצלעך פאר chatbots, רעקאמענדאציע-סיסטעמען, knowledge graphs און זוך-לייזונגען פאר פירמעס.

מען ניצט עס אויך אין פירמע-וויסן פארוואלטונג, זוך-מאשינען מיט AI, אויספארשן געזעצן און compliance, און מעדיצינישע AI. עס ווערט ערקלערט אז דורך מולטימאדאל embedding און וועקטאר-אפהיטונג קענען

ביזנעסער פארבעסערן ווי זיי זוכן, ווי גוט זייערע AI מאדעלן ארבעטן, און קלוגער באשליסן אין פארשידענע אינדוסטריעס.

דעמא: א chatbot פאר ווידעאס און בילדער מיט Gradio (17:22:06)

אין דער דעמא ווערט געוויזן ווי אזוי צו פארבינדן Gradio, א מולטימאדאל RAG ליניע, און ווידעא-פאראַרבעטונג, כדי צו בויען א chatbot וואס קען ארבעטן מיט ווידעאס און בילדער. מען הייבט אן מיט דער שורה:

```
from gradio_utils import get_demo
```

די טעקע gradio_utils.py אנטהאלט דעם גאנצן קאוד וואס מען נעמט אריין. אין קורצן טוט זי דאס:

- זי נעמט אריין ביבליאטעקן ווי Gradio, LanceDB, LangChain, MoviePy, PIL און נאך;
- זי פאראַרבעט ווידעאס דורך ארויסנעמען בילדער (frames) און שאפן א שריפטלעכן טעקסט (transcript);
- זי ניצט LanceDB אלס vector store פארן אויסזוכן, און מאכט embeddings מיט BridgeTower embeddings;
- זי שטעלט אויף א שמועס-AI סיסטעם וואס פארבינדט ענטפערס פון אויסזוכן מיט א שפראך-מאדעל;
- זי בויט א וועב-אינטערפעיס מיט Gradio, וואו באניצער שרייבן זייערע פראגעס און רעדן מיטן chatbot ווען א ווידעא און א בילד.

דערנאך גייט מען דורך די אימפארטן איינס נאך איינס:

- Gradio: שאפט א וועב-אינטערפעיס פארן רעדן מיטן chatbot;
- pathlib: ארבעט מיט טעקעס; os: נאך טעקע-אפעראציעס;
- enum, dataclasses און typing: צו באשטימען סטרוקטורירטע אביעקטן אין Python;
- PredictionGuard: א מעטאד פאר inference;
- BridgeTower embeddings: מאכט embeddings פון טעקסט, בילדער און סטרוקטורירטע דאטא אין א vector store;
- MultimodalLanceDB: א וועקטאר-דאטאבאזע פארן אויסזוכן;
- LVLM: א גרויסער וויזועל-שפראך מאדעל, וואס פארשטייט טעקסט צו בילד און בילד צו טעקסט (עס ווערט צוגעזאגט אז די זאכן וועלן באזונדער ערקלערט ווערן אין ווייטערדיגע מאדולן);
- MoviePy: פאראַרבעט ווידעאס.

דערנאך זעט מען א טייל פאר באהאנדלען טעותים: אן אלגעמיינע טעות-מעלדונג, וואס קומט ארויף אויב עפעס גייט נישט ווי עס דארף אין דער ליניע.

צום סוף הייבט מען אן מיט ווידעא-פאראַרבעטונג: ארויסנעמען א קורץ שטיקל פון א ווידעא ביי א געוויסן צייט-פונקט. דאס שטיקל ווערט אפגעהיט אין א טעקע פאר צעטיילטע ווידעאס, און מען ניצט MoviePy צו שניידן דאס שטיקל און זיכער מאכן אז דער קול שטימט מיטן בילד.

א מולטימאדעל RAG טשעטבאט און ווידעא-פאראַרבעטונג (17:25:44)

דער RAG-קייט און דער טשעטבאט-סטרוקטור (17:25:44)

אין דעם טייל פונעם קאוד ווערט אויפגעשטעלט דער RAG-קייט (RAG chain). דער קורס ערקלערט וואס יעדער טייל טוט: ערשטנס ווערט איניציאליזירט LanceDB אלס "וועקטאר-סטאר", א דאטא-באזע וואו מען האלט די AI-עמבעדינגס. צום עמבעדן ווערט גענוצט BridgeTower embeddings, וואס קענען עמבעדן טעקסט, בילדער און ווידאוס. דערנאך ווערט אויפגעשטעלט א retriever וואס זוכט ארויס די מערסטע פאסיגע פרעימס (בילדלעך) פונעם ווידעא.

עס ווערט געגעבן א ביישפיל: דער ווידעא וואס ווערט אריינגעברענגט איז פון א ספעיס-סטאנציע. אויב מען שרייבט דעם טעקסט "spacewalk", גיט די סיסטעם צוריק די פרעימס אדער די עטלעכע סעקונדעס פונעם ווידעא וואו מען זעט ווי א מענטש גייט טאקע אינדרויסן אין קאסמאס ("ספעיס-וואק"). אויסערדעם ווערט אריינגעבויט א שפראך-מאדעל, LVLM, וואס שאפט די ענטפערס.

דערנאך ווערט דעפינירט די פראמפט-פאראַרבעטונג. דער גאנצער סיסטעם איז אין גרונט א טשעטבאט, און ביי יעדן פראמפט טוט ער:

- ער נעמט ארויס פונעם וועקטאר-סטאר דעם מערסטן פאסיגן ווידעא-שטיק.
- ער נעמט ארויס דעם טראנסקריפט, דעם פרעים און די מעטאדאטא.
- ער גיט צוריק די דאטא פארמאטירט, גרייט פאר ווייטערדיגער AI-פאראַרבעטונג.

עס ווערט באמערקט אז אין דעם דעמא וועט מען זען צוויי טיילן אין יעדן ענטפער: א טעקסט וואס ענטפערט אויף דער פראגע, און דאס ספעציפישע ווידעא-שטיק פון וואו דער ענטפער קומט.

ווייטער איז דא אן אינסטאנץ פונעם AI-טשעטבאט וואס פירט די שמועס-היסטאריע. ער האלט די היסטאריע פונעם שמועס, די ווידעא-דאטא און די רעזולטאטן פון דער AI-אויסזוכונג, און ער פירט חשבון פון די בילדער, די קעפלעך (captions) און די אַרויסגענומענע ווידעא-שטיקער.

נאך ווייטער ווערט געשאפן א וועב-אינטערפעיס מיט Gradio, וואס האט:

- א ווידעא-שפילער צו זען די ארויסגענומענע ווידעא-שטיקער;
- א טשעטבאט פאר די AI-ענטפערס אין שמועס-פארם;
- א drop-down מעניו מיט פאראויס-צוגעגרייטע פראגעס;
- "Clear" און "Submit" קנעפלעך פארן באניצער.

מען זעט אין קאוד אויך עטלעכע בדיקות: די לענג פונעם איצטיגן שמועס דארף זיין צוויי, ד.ה. נאר צוויי שורות, און דער ערשטער ענטפער-מעסעדזש פונעם איצטיגן שמועס דארף זיין None.

צוזאמענגענומען ערקלערט דער קורס אז דער סקריפט בויט א מולטימאדעל AI-טשעטבאט. ער נעמט ארויס פרעימס און טראנסקריפטן פון א ווידעא און האלט די עמבעדינגס אין א וועקטאר-דאטא-באזע. ער זוכט ארויס דעם פאסיגן ווידעא-אינהאלט לויט דער פראגע פונעם באניצער, שאפט ענטפערס מיטן שפראך-מאדעל LVLM, און גיט א Gradio-אינטערפעיס וואו דער באניצער קען ארבעטן מיט די ווידאוס און מיטן טשעטבאט. אזא סיסטעם פאסט, לויטן קורס, באזונדערס פאר ווידעא-זוכן מיט AI, ארויסנעמען אינהאלט, און אינטעראקטיוו לערנען.

לאנצירן די אפליקאציע (17:28:47)

איצט לאזט מען לויפן דעם קאוד. ערשט שרייבט מען debug = False. עס ווערט באמערקט אז מען קען עס טוישן אויף True וועז מען וויל זוכן א פעלער. דערנאך רופט מען אויף די פונקציע get_demo און לאנצירט:

get_demo() = demo

(server_port=9999, debug=debug ,"0.0.0.0"demo.launch(server_name=

עס ווערט ערקלערט אז get_demo איז די פונקציע וואס מען האט פריער געזען, וואס גיט צוריק דעם demo-אביעקט. נאכן לויפן לויפט די אפליקאציע אויף א לאקאלן URL, און מען קען אויך שאפן א פובליק לינק.

דער דעמא: פראגעס אויפן ספעיס-סטאנציע ווידעא (17:30:20)

די אפליקאציע עפנט זיך, א Gradio-אינטערפעיס (אונטן שטייט "Built with Gradio"). מען קען אריינשרייבן א פראגע, און עס זענען דא פאראויס-צוגעגרייטע פראגעס.

- ערשטע פראגע: "astronaut spacewalk". מען דריקט Send, און עס נעמט א ווייל. דער ענטפער: אויפן בילד זעט מען א אסטראנויט אין א ווייסן ספעיס-אנצוג, וואס שטייט אויף א ספעיס-שיף און טוט א ספעיס-וואק. וועז מען שפילט דעם ווידעא, זעט מען אז עס איז נישט דער גאנצער ווידעא. די סיסטעם האט פונעם גאנצן ווידעא ארויסגענומען נאר דאס שטיקל וואו דער מענטש טוט דעם ספעיס-וואק.
- מען קען דריקן "Clear History" אויב מען וויל נישט אז דער טשעטבאט זאל געדענקען די פריערדיגע שמועס.
- צווייטע פראגע: "What is the name of one of the astronauts?" נאכן אויסמעקן די היסטאריע און שיקן די פראגע, ענטפערט דער טשעטבאט אז איינער פון די אסטראנויטן הייסט) Robertדער פאמיליע-נאמען ווי עס הערט זיך אין דער אויפנאמע: "Benin"). וועז מען שפילט דאס ווידעא-שטיקל, זעט מען דעם נאמען אויפן עקראן.
- מען קען אויך שרייבן אן אייגענע פראגע, און מען קען אויך ארויפלאדן אן אייגענעם YouTube ווידעא.
- דריטע פראגע: "What does the astronaut say?" דער ענטפער ציטירט דעם אסטראנויט בערך אזוי: "די אויסזיכט איז אלעמאל וואונדערליך. איך האב נישט געמיינט אז איך וועל טון נאך א ספעיס-וואק, און איצט האב איך די געלעגנהייט". עס ווערט באמערקט אז דער אודיא איז דערווייל נישט פאראן אינעם שטיקל, אבער דער ענטפער איז ריכטיג.

דער קורס שליסט אפ אז אזוי שאפט מען אן אינטעראקטיוון דעמא, און אזוי לערנט מען די ארכיטעקטור פון א מולטימאדעל RAG סיסטעם.

נייער מאדול: פאר-פאראַרבעטן ווידעאס פאר מולטימאדעל RAG (17:32:56)

דער נעקסטער מאדול לערנט ווי אזוי מען פאראַרבעט ווידעאס פאראויס פאר א מולטימאדעל RAG. ערשט טוישט מען דעם טיטל פונעם נאטבוק אויף "Multimodal RAG Video Processing". דערנאך אימפארטירט מען די הויפט-ביבליאטעקן:

- from pathlib import Path
- import os, און from os import path as osp
- import json, import cv2, import webvtt, import whisper

- from moviepy.editor import VideoFileClip
- from PIL import Image
- import base64

ביים ערשטן לויפן איז געווען א פעלער, ווייל א פאר אותיות זענען געווען גרויס און דארפן זיין קליין. נאכן פארריכטן ארבעט עס. דער קורס ערקלערט יעדע ביבליאטעק:

- pathlib.Path: גיט אן אביעקט-אריענטירטן אופן צו האנדלען מיט פיילן-סיסטעם וועגן (paths). מען נוצט עס צו שאפן, פירן און באשטעטיגן פיילן-וועגן גרינג.
- os: רעדט מיט דער אפערירונג-סיסטעם צו טון ארבעטן מיט פיילס און דירעקטאריעס. עס העלפט פירן פיילס, דירעקטאריעס און environment-ווריעבלס.
- os.path (קורץ osp): א אונטער-מאדול פון os מיט פונקציעס צו באארבעטן פיילן-וועגן. זייער ניצלעך צו פארבינדן, נארמאליזירן און ארויסנעמען טיילן פון א פיילן-וועג.
- json: א מאדול צו ענקאדירן און דעקאדירן JSON-דאטא. מען נוצט עס צו האלטן, לייענען און שרייבן מעטאדאטא אין א סטרוקטורירטן פארמאט.
- cv2: דאס איז OpenCV, אן אפן-קוואל ביבליאטעק פאר "קאמפיוטער-זעאונג" און בילד-פאראַרבעטונג. מען נוצט עס צו ארויסנעמען פרעימס פון ווידעאס און צו באארבעטן בילדער.
- webvtt: א ביבליאטעק צו האנדלען מיט WebVTT סובטיטלען (VTT). מען נוצט עס צו ארויסנעמען און באארבעטן סובטיטלען פון ווידעא-פיילס.
- whisper: אן AI-מאדעל פון OpenAI פאר אויטאמאטישע רייד-דערקענונג. עס פארוואנדלט די רייד אינעם ווידעא אין טעקסט, פאר טראנסקריפציע און אנאליז.
- moviepy.editor.VideoFileClip: א מאדול צו רעדאגירן און באארבעטן ווידעאס, למשל אפשניידן, ארויסנעמען דעם אודיא, און פאראַרבעטן ווידעא-פיילס.
- PIL.Image: PIL (Pillow) איז א ביבליאטעק פאר בילד-פאראַרבעטונג. עס ערמעגלעכט צו טוישן די גרייס, אפהיטן און מאדיפיצירן בילדער גרינג.
- base64: א מאדול צו ענקאדירן און דעקאדירן base64-דאטא. מען נוצט עס צו פארוואנדלען בילדער און ווידעאס אין א טעקסט-פארמאט, כדי מען זאל זיי גרינג קענען אפהיטן און איבערשיקן.

צוזאמען, ערקלערט דער קורס, זיכערן די אימפארטן אן עפעקטיוע ווידעא-פאראַרבעטונג פאר מולטימאדעל RAG. אזוי קענען AI-מאדעלן פאראַרבעטן און ארויסזוכן פאסיגן אינהאלט פון ווידעא, אודיא און בילדער.

אראפלאדן די ווידעאס (17:38:08)

דער נעקסטער שריט איז אראפלאדן א "קארפוס" ווידעאס. מען נוצט א YouTube ווידעא, אבער יעדער קען אויסקלייבן זיין אייגענעם און אריינלייגן דעם לינק. דער קאוד אינעם Jupyter נאטבוק לאדט אראפ דעם YouTube ווידעא מיט זיינע סובטיטלען און לייגט זיי אין א געוויסער דירעקטאריע. דערנאך ווייזט ער אן דעם פיילן-וועג און דעם אינהאלט פונעם טראנסקריפט. מען נוצט דערפאר צוויי פונקציעס פונעם utils מאדול:

- download_video: לאדט אראפ YouTube ווידעאס.
- get_transcript_vtt: נעמט ארויס די סובטיטלען (דעם טראנסקריפט) פונעם YouTube ווידעא אין VTT-פארמאט.

דערנאך דרוקט מען אויס צוויי פייל-וועגן: דעם פונעם ווידעא און דעם פונעם טראנסקריפט (סובטיטלען). מען זעט אז דער ווידעא-פייל הייסט "Welcome back to Planet Earth", און דער צווייטער וועג איז דער פון די קעפלעך, ד.ה. דעם טראנסקריפט.

דערנאך לאזט מען לויפן א shell-באפעל:

{vid1_transcript_filepath} n15- head!

עס ווערט ערקלערט אז head מיט 15 ווייזט די ערשטע 15 שורות פונעם טראנסקריפט-פייל, און דער נאמען אין די געוווּנדענע קלאמערן ווערט דינאמיש אויסגעטוישט מיטן אמת'ן פייל-וועג.

אראפלאדן א צווייטן ווידעא פון Amazon S3 (17:41:16)

איצט אימפארטירט מען from urllib.request import urlretrieve, און מען לייגט צו א צווייטן ווידעא. דער קאוד לאדט אראפ א ווידעא פון אן Amazon S3 URL, לייגט עס אין א לאקאלער דירעקטאריע, און זיכערט אז די נויטיגע פאלדערס עקזיסטירן. דער צווייטער ווידעא הייסט "toddler in playground" (א קליין קינד אויף א שפילפלאץ). די פרטים:

- urlretrieve: א פונקציע פון urllib.request ס'Python מאדול, וואס לאדט אראפ א פייל פון א געגעבענעם URL און לייגט עס לאקאל.
- דער URL פונעם צווייטן ווידעא איז צעטיילט אויף צווי שורות, כדי עס זאל זיין גרינגער צו לייענען.
- מען באשטימט דעם פאלדער וואו מען וויל אפהיטן דעם ווידעא.
- mkdir(parents=True, exist_ok=True)... שאפט די דירעקטאריע אויב זי עקזיסטירט נישט. parents=True זיכערט אז אלע מיטל-דירעקטאריעס ווערן געשאפן אויב זיי פעלן, און exist_ok=True פארמיידט א פעלער אויב די דירעקטאריע עקזיסטירט שוין.
- urlretrieve(vid2_url, osp.join(...))[0] לאדט אראפ דעם פייל צום באשטימטן פאלדער, און דער [0] זיכערט אז מען האלט אינעם ווערייעבל פונעם צווייטן ווידעא נאר דעם פייל-וועג פונעם אראפגעלאדענעם ווידעא.

הילפס-פונקציעס פון utils (17:43:22)

דערנאך אימפארטירט מען הילפס-פונקציעס פונעם utils פייל, וואס איז שוין געוויזן געווארן אינעם פריערדיגן מאדול. מען אימפארטירט א פונקציע פאר צייט-פארוואנדלונג (str2time) און maintain_aspect_ratio_resize. ביים ערשטן לויפן קומט א פעלער "cannot import name". נאכן פארריכטן ארבעט עס.

דער קורס ערקלערט דעם צוועק:

- די צייט-פונקציע פארוואנדלט "טיים-סטעמפס" אין סעקונדעס אדער מיליסעקונדעס. למשל 00:02:15)צוויי מינוט און פופצן סעקונדעס) ווערט פארוואנדלט אין סעקונדעס אדער מיליסעקונדעס. מסתמא וועט מען עס נוצן צום ארויסנעמען ווידעא-שטיקער, וואו מען דארף פארארבעטן טיים-סטעמפס.
- maintain_aspect_ratio_resize זיכערט אז בילדער ווערן פארקלענערט אדער פארגרעסערט אן פארקרימען זייער פראפארציע. מסתמא וועט מען עס נוצן צו פארארבעטן ווידעא-פרעימס אדער צו שאפן קליינע פאר-בילדלעך (thumbnails) אינעם מולטימאדעל RAG סיסטעם.

צום סוף פון דעם אפשניט הייבט מען אן שרייבן א פונקציע וואס נעמט ארויס און היט אפ פרעימס און מעטאדאטא. זי באקומט אלס אינפוט א ווידעא מיט זיין טראנסקריפט. די ערקלערונג פונעם צוועק פון דער פונקציע הייבט זיך אן גלייך דערנאך.

ווִידעא-פראצעסירונג און וועקטאר-סטאָרס אין מולטימאדאל RAG (17:45:31)

א פונקציע וואס נעמט ארויס בילדער פון א ווִידעא מיט זייער טעקסט (17:45:31)

דער קורס ערקלערט א פונקציע וואס נעמט ארויס "פרעימס" (איינצלנע בילדער) פון א ווִידעא, און פארבינדט יעדן בילד מיט די טעקסטועלע מעטא-דאטא פונעם טראנסקריפט פון דעם ווִידעא.

די פונקציע באקומט פיר פאראמעטערס:

- path to video – דער וועג צום ווִידעא-פייל.
- path to transcript – דער וועג צום סובטיטל-פייל, וואס קומט אין VTT פארמאט.
- path to save extracted frames – דער דירעקטאריע וואו די ארויסגענומענע בילדער וועלן אפגעהיט ווערן.
- path to save metadatas – דער דירעקטאריע וואו דער מעטא-דאטא פייל וועט אפגעהיט ווערן.

אינעווייניג ווערט געמאכט א ליידיגע ליסטע metadatas, וואס וועט האלטן א דיקשאנערי מיט מעטא-דאטא פאר יעדן ארויסגענומענעם פרעים. דערנאך ווערט דער ווִידעא געלאדנט מיט cv2, ד.ה. OpenCV, און דער טראנסקריפט ווערט געלאדנט מיט דער ביבליאטעק webvtt.

דערנאך קומט דער לופ: for idx, transcript in enumerate(trans). ער גייט איבער יעדן איינטראג אין סובטיטל-פייל. יעדער איינטראג שטעלט פאר א געוויסן שטיקל פונעם ווִידעא, מיט א אנהייב-צייט, א סוף-צייט און דעם טעקסט-קעפשן.

די שריט אין יעדן איינטראג:

- מען רעכנט אויס א "מיטל-צייט" (mid time), דאס איז דער מיטלסטער צייט-שטעמפל צווישן אנהייב און סוף. די אנהייב- און סוף-צייטן ווערן פריער איבערגעמאכט אויף מיליסעקונדעס מיט דער פונקציע str2time. דער פרעים ווערט שפעטער ארויסגענומען פונעם מיטל-פונקט.
- מען רייניגט אויס דעם טראנסקריפט: מען נעמט אוועק די ניי-שורה צייכנס (\n).
- מען נעמט ארויס א פרעים פון יענער פאזיציע אינעם ווִידעא.
- אויב עס איז געלונגען (if success), ווערט דער פרעים פארקלענערט (resize) און אפגעהיט ווי א JPEG בילד.
- מען לייגט אריין די מעטא-דאטא: דעם וועג צום בילד-פייל, דעם טראנסקריפט-טעקסט, די segment ID, דעם וועג צום ווִידעא, און די mid time, ד.ה. דעם צייט-שטעמפל פונעם ארויסגענומענעם פרעים.
- פאר פעלערן: אויב מען קען נישט ארויסנעמען דעם פרעים, ווערט געדרוקט א פעלער-מעלדונג "error cannot extract".

צום סוף ווערט די גאנצע מעטא-דאטא אפגעהיט ווי א JSON פייל, metadata.json, אינעם געגעבענעם וועג.

לויפן די פונקציע און קוקן אויף די רעזולטאטן (17:48:38)

מען לויפט דעם קאוד, און ער גייט דורך אן פעלער. דערנאך ווערן באשטימט די וועגן וואו מען וויל אפהיטן די ארויסגענומענע פרעימס און זייער מעטא-דאטא. מען שאפט די אויסגאנג-פאלדערס אויב זיי עקזיסטירן נאך נישט, און מען רופט די פונקציע.

מען דרוקט ארויס אן איינטראג פון) metadatasמען דארף אכטונג געבן אז דער נאמען איז metadatas). מען זעט אין רעזולטאט דעם צייט-שטעמפל, דעם וועג צום ווידעא, דעם ווידעא-סעגמענט, און די אנדערע פעלדער.

א ווידעא אן א טראנסקריפט: Whisper (17:49:08)

דערנאך ווייזט דער קורס ווי אזוי צו פראצעסירן א ווידעא-קארפוס וואס האט נישט קיין פארטיגן טראנסקריפט. מען נעמט ארויס דעם קול (audio) פונעם ווידעא. דער קורס ווארנט אז דער פראצעס נעמט געווענליך לאנג, בערך 1 ביז 2 מינוט, לויט וויפיל זכרון (memory) עס האט די סיסטעם. פאר בעסערע רעזולטאטן קען מען פרובירן גרעסערע Whisper מאדעלן, ווי למשל large-v2.

נאכדעם וואס דער קול איז ארויסגענומען, שרייבט מען:

- model = whisper.load_model("small"). דאס לאדנט די קליינע ווערסיע פון OpenAI'ס Whisper מאדעל. Whisper איז אן AI מאדעל פאר אויטאמאטישע רייד-דערקענונג, אויף ענגליש automatic speech recognition (ASR).
- options = א דיקשאנערי מיט task = "translate", best_of = 1, און language = "en" (ענגליש).
- results = model.transcribe(path_to_extracted_audio_file, options).

עס ווערט ערקלערט וואס יעדע איינשטעלונג טוט. task = "translate" שרייבט אויף דעם קול און איבערזעצט נישט-ענגלישע רייד אויף ענגליש. best_of = 1 הייסט אז מען פרובירט נאר איין מאל אויפצושרייבן. העכערע צאלן פארבעסערן די גענויקייט, אבער דער פראצעס נעמט לענגער. results לויפט דעם Whisper מאדעל אויפן ארויסגענומענעם קול און שאפט א טראנסקריפט אויף ענגליש.

אפהיטן דעם טראנסקריפט אין VTT פארמאט (17:52:20)

דער ווייטערדיגער קאוד מאכט איבער דעם טראנסקריפט אויף VTT פארמאט, וואס מען ניצט פאר ווידעא-סובטיטלען. ער זארגט אז די צייט-שטעמפלען זאלן זיין ריכטיג פארמאטירט, כדי דער טעקסט זאל גיין גלייך מיטן בילד. ער שאפט א וועג פאר דעם נייעם סובטיטל-פייל, און היט אפ דעם פייל אין שרייב-מאדע, דערפאר שטייט דארט "w". צום סוף ווייזט מען די ערשטע פאר שורות פונעם געשאפענעם פייל. מען זעט די סובטיטלען מיט זייערע צייט-שטעמפלען.

LVLM: א מאדעל וואס באשרייבט א בילד (17:52:51)

דערנאך ווערט באשטימט אן LVLM, דאס הייסט large vision language model. ער שאפט א קעפשן (באשרייבונג) פאר א בילד וואס איז ארויסגענומען געווארן פון א געוויסן ווידעא. ערשט נעמט מען ארויס דאס בילד פונעם ווידעא, און דערנאך שאפט דער מאדעל א באשרייבונג פאר דעם בילד.

מען שטעלט א טעקסט-פראמפט: "Can you describe the image?". דער מאדעל וועט אנאליזירן דאס בילד און שאפן א קעפשן.

דערנאך לאדנט מען א ארויסגענומענעם פרעים:

- path_to_frame = os.path.join(...) מיטן ווידְעא-דירעקטאריע און דעם נאמען פונעם פרעים-פייל, א jpg.
- frame = Image.open(path_to_frame), וואס עפנט דאס בילד מיטן PIL מאדול Image.
- מען שרייבט frame, און דאס בילד ווערט געוויזן אין Jupyter Notebook וועו מען לויפט דעם צעל.

עס ווערט ערקלערט אז דאס איז דער וועג צום פינפטן ארויסגענומענעם פרעים. מען לויפט עס, און מען זעט דאס בילד וואס איז ארויסגענומען פונעם ווידְעא.

פרעגן דעם מאדעל וואס עס איז אויפן בילד (17:55:31)

יעצט פרעגט מען דעם מאדעל וואס דאס בילד איז:

- from utils import lvlm_inference, encode_image
- encode_image(path_to_frame) = image
- lvlm_inference(lvlm_prompt, image) = caption, און דערנאך print(caption).

ביים לויפן קומט ארויס א פעלער: "lvlm_prompt is not defined". דער קורס ערקלערט די סיבה: פריער האט מען דעם פראמפט אפגעהיט אונטער א נאמען וואס איז נישט lvlm_prompt. נאכן פארענדערן און וויידער לויפן, ארבעט עס.

דער מאדעל באשרייבט דאס בילד אזוי: עס ווייזט א ספּעיס-שאטל (space shuttle), מיט א מענטש וואס שוועבט אין דער לופט. די שאטל שטייט אין מיטן פונעם בילד, און דער מענטש זעט אויס ווי ער ארבעט אויפן פלי-שיף. דער קורס זאגט אז דאס איז די קראפט פון מולטימאדאל RAG.

ארויסנעמען פרעימס און מעטא-דאטא מיט LVLM (17:57:35)

דערנאך ווערט געשריבן נאך א פונקציע, כמעט די זעלבע ווי די ערשטע. זי נעמט ארויס פרעימס און מעטא-דאטא פון ווידְעאס, ניצנדיג LVLM inference. די ליסטע metadatas האלט די מעטא-דאטא פון אלע ארויסגענומענע פרעימס. מען לאדנט דעם ווידְעא מיט cv2, מען נעמט די פרעימס, מען גייט איבער זיי אין א לופ, היט זיי אפ ווי JPEG, גרייט צו די מעטא-דאטא, און היט אלעס אפ.

דערנאך באשטימט מען די וועגן פאר די פרעימס, די מעטא-דאטא און די טראנסקריפטן. מען שאפט די פאלדערס אויב זיי זענען נישטא, און מען רופט די פונקציע.

אינעם רעזולטאט באשרייבט דער מאדעל: א יונגער בחור גייט אויף א שפילפלאץ, האלט א וואסער-פלאש אין דער האנט, און טראגט א בלויע העמד.

דער קורס פאסט צוואם: אזוי פראצעסירט מען ווידְאס פאר מולטימאדאל ארבעט. מען קען ארויסנעמען א געוויסן בילד וואס מען ווייל, בעטן דעם מאדעל עס צו באשרייבן, און ווייטער ארבעטן דערמיט ווי מען ווייל. אין דעם קומענדיגן מאדול קומען וועקטאר-סטאָרס אין מולטימאדאל RAG.

וואס זענען וועקטאר-סטאָרס אין RAG? (17:59:46)

דער נייער מאדול רעדט פון מולטימאדאל אויסזוכן (retrieval) פון א וועקטאר-סטאָר. וועקטאר-סטאָרס שפילן א הויפט-ראלע אין RAG. זיי לאזן אפהיטן און שנעל אויסזוכן "עמבעדינגס" (embeddings) מיט אסאך דימענסיעס, פאר טעקסט, בילדער און מולטימאדאל דאטא. אין RAG נעמט א "retriever" די פאסיגע אינפארמאציע פונעם וועקטאר-סטאָר, און א "generator", ווי אן LLM, ניצט זי אויסצושטעלן א באלערנטן

ענטפער. אין מאדול 2 האט מען שוין געלערנט פון וועקטאר-סטאָרס, אבער מען גייט נאכאמאל שנעל איבער ווי זיי ארבעטן:

- עמבעדינג-שאפונג: דאטא אין פארעם פון טעקסט, בילדער אדער ווידֵעא-פרעימס ווערט איבערגעמאכט אויף געדיכטע וועקטאר-פארשטעלונגען, מיט מאדעלן ווי די פון OpenAI אדער CLIP.
- אפהיטונג: די עמבעדינגס, צוזאמען מיט מעטא-דאטא, ווערן אפגעהיט אין א וועקטאר-דאטאבייס.
- אויסזוכן: ווען מען שטעלט א פראגע, געפינט דער וועקטאר-סטאָר עמבעדינגס וואס זענען ענליך אין באדייט. מען ניצט דערפאר טעכניקן ווי K nearest neighbors (KNN) אדער approximate nearest neighbor (ANN).
- פארשטארקן דעם LLM: די געפונענע רעזולטאטן גיבן קאנטעקסט צום גענעראטיוון מאדעל, און דאס פארבעסערט די גענויקייט און די שייכות פון די ענטפערס.

פארוואס וועקטאר-סטאָרס זענען וויכטיג:

- זיי זוכן זייער שנעל ענלעכקייטן, פאר AI אפליקאציעס וואס ארבעטן אין רעאלער צייט.
- זיי פארבעסערן LLMs דורכן באזירן די ענטפערס אויף די געפונענע אינפארמאציע.
- זיי זענען גרייט צו וואקסן (scalable) און מולטימאדאל: זיי ארבעטן מיט טעקסט, בילדער און ווידֵעא.

דערפאר זענען זיי קריטיש אין מאדערנע AI אפליקאציעס, ווי טשעט-באטס, זוך-מאשינען, רעקאמענדאציע-סיסטעמען, און נאך.

דער סעט-אפ: אימפארטירן די ביבליאטעקן (18:02:04)

מען הייבט אן מיטן סעט-אפ און אימפארטירט די הויפט-ביבליאטעקן:

- from mm_rag.embeddings.bridgetower_embeddings import BridgeTowerEmbeddings (mm מיינט מולטימאדאל).
- from mm_rag.vectorstores.multimodal_lancedb import MultimodalLanceDB
- אימפארטן פון os, json, lancedb, און from PIL import Image.
- פון utils א פונקציע פאר JSON פיילס, און א פונקציע display_retrieved_results.

מען לויפט דעם קאוד, און ער גייט דורך אן פעלער. דער קאוד אימפארטירט די נויטיגע ביבליאטעקן פאר עמבעדינגס, אפהיטונג, JSON און בילד-פראצעסירונג אין א מולטימאדאל RAG פייפליין. דער קורס ערקלערט וואס יעדע ביבליאטעק טוט:

- BridgeTowerEmbeddings: א מאדעל וואס מאכט איבער בילדער און טעקסט אויף וועקטאר-עמבעדינגס. מען ניצט אים פאר מאדעלן וואס זוכן אויס פאסיגע בילדער און טעקסט אין א וועקטאר-דאטאבייס.
- MultimodalLanceDB: א וועקטאר-דאטאבייס פאר אפהיטן און אויסזוכן מולטימאדאל עמבעדינגס. ער שטיצט עמבעדינגס פון בילדער, טעקסט און סטרוקטורירטע דאטא. LanceDB איז די ביבליאטעק אונטן, וואס היט אפ די עמבעדינגס אויף אן עפעקטיוון אופן, פאר שנעלן זוכן ענלעכקייטן.
- json: לייענט און שרייבט JSON פיילס אין א סטרוקטורירטן פארמאט, ווי מעטא-דאטא און עמבעדינגס.
- os: גיט פונקציעס פאר ארבעטן מיט פיילס און דירעקטאריעס.

- PIL: איז פאר לאדנען און פראצעסירן בילדער.

מולטימאדעל זוכן מיט LanceDB און LVLM מאדעלן (18:06:13)

די אימפארטן און זייער ציל (18:06:13)

די לעקציע הייבט אן מיט ערקלערן די לעצטע אימפארטן פון דעם נאטבוק. איינע פון זיי איז פאר ווייזן בילדער, און זי איז זייער נוצלעך פאר ענדערן די גרייס פון בילדער איידער מען מאכט פון זיי embeddings . די שורה from utils import load_json_file דינט צו לאדן און צעלייגן JSON מעטא-דאטא. די פונקציע display_retrieved_results ווייזט די געפונענע זוך-רעזולטאטן אין א סדר'דיגע פארעם, און מען ניצט זי מסתמא כדי צו זען די רעזולטאטן וואס קומען צוריק פון דער וועקטאר-דאטאבאזע.

עס ווערט סוממירט דער ציל פון די אלע כלים: מאכן embeddings פון בילדער און טעקסט פאר AI-באזירטע זוכענישן, אפהאלטן די embeddings, און ארבעטן מיט מולטימאדעל זוך-רעזולטאטן.

אויפשטעלן LanceDB אלס וועקטאר-סטאר (18:06:45)

דער נעקסטער שריט איז אויפשטעלן א LanceDB וועקטאר-סטאר. מען שרייבט צוערשט א ווערייאבל פאר דעם פייל-פאד, ווען די דאטאבאזע וועט ליגן, אונטער א shared_data פאלדער. דערנאך דעקלערירט מען דעם נאמען פון דער טאבעלע, און מען גיט איר דעם נאמען test.

נאכדעם איניציאליזירט מען דעם וועקטאר-סטאר מיט א שורה ענלעך צו:

lancedb.connect(LANCEDB_HOST_FILE) = db

ביים ערשטן לויפן קומט א טעות אז דער host file איז נישט דעפינירט. דער גורם איז געווען אז געוויסע אותיות האבן געדארפט זיין גרויסע, און נאך דעם תיקון לויפט עס.

עס ווערט ערקלערט וואס יעדער טייל טוט:

- דער host file באשטימט ווען דער LanceDB דאטאבאזע-פייל ליגט.
- דער טאבעלע-נאמען איז test.
- די connect פונקציע פארבינדט צום LanceDB פייל, און אויב די דאטאבאזע עקזיסטירט נאך נישט, ווערט זי באשאפן.

אריינלאדן דעם ווידעא-קארפוס מיט LangChain (18:08:52)

יעצט גרייט מען צו די דאטא, כדי אריינצולאדן דעם ווידעא-קארפוס אין LanceDB דורך LangChain. דער ערשטער טייל לאדט די מעטא-דאטא פיילס. מען דעפינירט די פייל-פעדער פון די JSON מעטא-דאטא פאר צוויי ווידעאס, video1 און video2. די פונקציע load_json_file לייענט די מעטא-דאטא פונעם JSON פייל אריין אין א Python dictionary. די מעטא-דאטא אנטהאלט די פעדער פון די ארויסגענומענע פריימס, די AI-געמאכטע קעפשאנס (דאס הייסט די טראנסקריפט), און נאך שייכותדיגע פרטים.

video1_trans איז די ליסטע פון קעפשאנס, דאס הייסט די טראנסקריפטן פון ווידעא איינס, און video1_img_path איז די ליסטע פון פעדער צו די ארויסגענומענע פריימס פון ווידעא איינס. ביי ווידעא צוויי איז עס פונקט אזוי.

צולייגן קאנטעקסט צו די טראנסקריפטן (18:09:54)

דערנאך דערהייבט מען די טראנסקריפט מיט קאנטעקסטואלע אינפארמאציע. מען שרייבט n = 7. די צאל באשטימט ווי־פיל שכנישע טראנסקריפט-שורות מען נעמט אריין ביי יעדן פריים, און דאס גיט מער קאנטעקסט ביים אפהאלטן אדער צוריקברענגען ווידעא-קעפשאנס.

דער קאוד פאר updated_video1_trans גייט דורך יעדע שורה אין video1_trans און פארבינדט דעם קעפשאן מיט n/2 פריערדיגע און n/2 שפעטערדיגע שורות. ער באהאנדלט אויך די עק-פעלער: ווען עס זענען נישט פארהאן גענוג פריערדיגע שורות, הייבט מען אן פונעם אינדעקס 0. אזוי באקומט יעדער פריים מער ארומיגן קאנטעקסט אין זיין טראנסקריפט. דערנאך ערהייבט מען די מעטא-דאטא: די אלטע טראנסקריפט אין video1 ווערט פארביטן מיט דער נייער, קאנטעקסט-באוואוסטער טראנסקריפט.

כדי צו זען דעם חילוק, דרוקט מען אויס א ביישפיל פון פאר און נאך דעם ענדערונג. מען שרייבט א print מיט "Transcript example before update", און דערנאך דרוקט מען די שורה מיט אינדעקס 6 פון video1_trans. נאכדעם קומט נאך א print, "after update", מיט [6]updated_video1_trans. ביים לויפן זעט מען אז די ערשטע ווערסיע איז א קורצע שורה וועגן דער ספעיס-שטאציע, אז מען האט איצט די גענויגנהייט עפעס צו טאן. נאך דעם ענדערונג איז דער טעקסט אסאך לענגער. ער נעמט אריין די שורות פון פאר און נאך, ענלעך צו: "צוויי חדשים, די אויסזיכט איז אלעמאל וואונדערליך, איך האב נישט געמיינט אז איך וועל נאכאמאל גיין אויף א spacewalk, און יעצט האב איך א געלעגנהייט..." און נאכדעם קומט נאך מער טעקסט. אזוי קען מען זיך איבערצייגן אז דער קאנטעקסט איז גענאטן צוגעלייגט געווארן.

אריינלייגן די דאטא מיט מולטימאדעל embeddings (18:13:34)

יעצט לייגט מען די דאטא אריין אין LanceDB מיט מולטימאדעל embeddings. מען ניצט BridgeTower embeddings. דאס איז א מאדעל וואס פארוואנדלט טעקסט און בילדער אין וועקטאר-embeddings , און די embeddings ווערן גענוצט פאר מולטימאדעל זוכעניש. די פראמעטערס זענען:

- text: פארבינדט די קעפשאנס פון ביידע ווידעאס.
- image_paths: פארבינדט די פעדער פון די ארויסגענומענע פריימס.
- embedding: ניצט BridgeTower embeddings צו מאכן די וועקטארן.
- metadatas: האלט אפ נאך פרטים.
- connection=db: פארבינדט צו LanceDB.
- table_name: ניצט די test טאבעלע אלס דאטאבאזע-טאבעלע.
- mode="overwrite": מאכט זיכער אז דער וועקטאר-סטאר ווערט אויסגעליידיגט פאר דעם אריינלייגן.

דערנאך לויפט מען דעם קאוד.

עפענען די טאבעלע און קוקן די שורות (18:14:38)

אינעם נעקסטן טייל טוט מען מולטימאדעל זוכעניש מיט LangChain. צוערשט שאפט מען א פארבינדונג צו דער טאבעלע אין LanceDB:

```
db.open_table(TBL_NAME) = tbl
```

דערנאך דרוקט מען אויס מיט א print f-string די צאל שורות אין דער טאבעלע, דורך tbl.to_pandas().shape[0]. עס ווערט באמערקט אז pandas איז א Python ביבליאטעק פאר data frames און shapes.

נאכדעם ווייזט מען די ערשטע דריי שורות: מען שרייבט tbl.to_pandas() מיט די קאלאמען text און image_path, און דערנאך .head. head(3) ווייזט די אויבערשטע שורות. ווען מען ניצט tail אנשטאט head, באקומט מען די לעצטע שורות. ביים לויפן זעט מען די ערשטע דריי שורות, מיט דעם טעקסט און דעם בילד-פאד פון יעדער שורה.

זוכן אינעם וועקטאר-סטאר מיט LangChain (18:16:44)

יעצט טוט מען די זוכעניש פונעם LanceDB וועקטאר-סטאר דורך LangChain. מען שאפט א וועקטאר-סטאר אין LanceDB מיט די פאלגנדע פראמעטערס:

- uri: באשטימט וואו די דאטאבאזע ליגט.
- embedding=embedder: ניצט BridgeTower embeddings פאר דער זוכעניש.
- table_name: באשטימט פון וועלכער טאבעלע מען נעמט די דאטא.

דערנאך שאפט מען א retriever וואס זוכט לויט ענלעכקייט (similarity), און ער טוט נישט מער ווי דאס.

מען פרעגט דעם וועקטאר-סטאר מיט דער פראגע "a toddler and an adult" (א קליין קינד און א דערוואקסענער). ביים לויפן קומט צוריק די מערסט שייכותדיגע אפגעהאלטענע דאטא, ביידע דער טעקסט און דאס בילד, און אויפן בילד זעט מען א דערוואקסענעם.

דערנאך ניצט מען similarity search מיט k=3. דאס גיט צוריק דריי ענלעכע בילדער מיט זייערע ענלעכע טעקסטן. ביים לויפן זעט מען דריי רעזולטאטן. ביי איינעם פון זיי קען מען נישט זיכער זען א דערוואקסענעם, אבער דאס קליינע קינד איז יא דארט. עס ווערט באטאנט אז דער רעזולטאט איז נאר "ענלעך", נישט פונקטליך. אזוי פארבעסערט מען די זוכעניש, ווייל מען ברענגט צוריק עטלעכע רעזולטאטן אנשטאט נאר איינעם.

וואס זענען LVLM, גרויסע וויזשן-שפראך מאדעלן (18:18:22)

א נייער מאדול הייבט זיך אן וועגן LVLMs, Large Vision Language Models, און ווי זיי העלפן ביי מולטימאדעל RAG סיסטעמען. LVLMs זענען AI מאדעלן וואס פארארבעטן און פארשטייען ביידע, וויזועלע און טעקסטועלע דאטא. נישט ווי די געווענליכע LLMs, וואס ארבעטן נאר מיט טעקסט, פאראיניגן LVLMs computer vision מיט (natural language processing) NLP. אזוי קענען זיי פארשטיין בילדער און וויידעאס צוואמען מיט טעקסטועלע באשרייבונגען.

ווי ארבעטן זיי:

- מולטימאדעל LVLMs :embedding פארוואנדלען ביידע, טעקסט און בילדער, אין א געמיינזאמען וועקטאר-ראום. דאס לאזט זיי פארגלייכן און פארבינדן די פארשידענע מאדאליטעטן.
- פאר-טריינירטע וויזשן-שפראך מאדעלן.
- cross-modal learning.

באקאנטע LVLMs:

- CLIP פון OpenAI: פאסט צו בילדער מיט זייערע מערסט שייכותדיגע קעפשאנס.
- BLIP-2: שאפט און פארשטייט באשרייבונגען אין נאטירליכער שפראך פאר בילדער.
- Flamingo: פאראַרבעט ביידע, בילדער און טעקסט, כדי צו ענטפערן אויף פראגעס.
- BridgeTower: וואס מען האט שוין פריער געוווצט. דאס איז א זייער עפעקטיוווער מולטימאדעל מאדעל פאר וויזשן-שפראך ארבעטן.

פארוואס זענען זיי וויכטיג? זיי מאכן מעגלעך מולטימאדעל זוכענישׂ, צי מען זוכט בילדער, טעקסט אדער וויידעאס. זיי פארבעסערן וווי AI נעמט אויף די וועלט, למשל ביים פארשטיין חכמה-בילדלעך (memes), טשארטס, טעקסט אדער געסקענטע דאקומענטן. זיי שטייען הינטער אפליקאציעס ווי image captioning, VQA און מולטימאדעל chatbots. בקיצור, LVLMs פארבינדן וויזשן מיט שפראך, און מאכן AI מער קאנטעקסט-באוואוסט און אינטעראקטיוו.

אויפשטעלן די אימפארטן פאר LVLM (18:20:56)

מען אימפארטירט די באזיס-ביבליאטעקן:

- from pathlib import Path: באהאנדלט פייל-פעדער און דירעקטאריעס.
- urllib.request urlretrieve: לאדט אראפ פיילס פון וועב, למשל בילדער און מאדעלן.
- PIL Image: לאדט און פאראַרבעט בילדער.
- IPython.display display: וויזוט בילדער אינעווייניג אינעם Jupyter נאטבוק.
- utils encode_image: פארוואנדלט א בילד אין base64 פארמאט פאר LVLM inference.
- utils prediction_guard_llava_conv: מסתמא א פריער-דעפינירטע שמועס-סטרוקטור פאר LVLM מאדעלן ווי LLaVA.
- lvlm_inference_with_conversation: טוט inference מיט אן LVLM מאדעל, און שטיצט א קאנטעקסטועלן שמועס מיט בילדער.

צוגרייטן די דאטא פאר מולטימאדעל פאראַרבעטונג (18:24:06)

דערנאך גרייט מען צו דאטא מיט פייל-פעדער און טעקסט-באשרייבונגען פאר דער מולטימאדעל זוכענישׂ. אין url1 האלט מען אפ דעם URL פון א בילד פונעם COCO דאטא-סעט, כדי עס אראפצולאדן און צו פאראַרבעטן פאר דער ארבעט. דערנאך דעפינירט מען די מעטא-דאטא פאר דעם ערשטן בילד: link איז דער URL פון בילד, transcript איז א טעקסט-באשרייבונג (דער alt text), און path_to_file איז דער לאקאלער פאד וואו דאס בילד ווערט אפגעהאלטן. img2_metadata איז די מעטא-דאטא פאר דעם צווייטן בילד, וואס איז ארויסגענומען פון א וויידעא-פריים, מיט א transcript און א path_to_file . ביים דריטן בילד, img3_metadata, איז עס די זעלבע זאך: א transcript און א path_to_file.

צום סוף קוקט מען צי דער בילד-פייל אויפן געגעבענעם פאד עקזיסטירט לאקאל. אויב נישט, לאדט מען אראפ דאס בילד פונעם link און מען האלט עס אפ אין path_to_file פון img1_metadata. אזוי מאכט מען זיכער אז די נויטיגע בילדער זענען פארהאן פאר דער פאראַרבעטונג. דערנאך לויפט מען דעם קאוד.

הקדמה צו LLaVA (18:26:10)

איצט גייט מען איבער צו LLaVA, Large Language and Vision Assistant. דאס איז א גרויסער מולטימאדעל מאדעל, טריינירט פון אנפאנג ביז סוף (end-to-end), וואס פארבינדט א vision encoder מיט אן LLM, פאר אלגעמיינעם וויזועלן און שפראכלעכן פארשטאנד. LLaVA זעט נישט נאר בילדער, נאר פארשטייט זיי. ער לייענט דעם טעקסט וואס איז אריינגעשריבן אין זיי, און טראכט איבער זייער קאנטעקסט, אלץ בשעת ער שמועסט מיטן באניצער אויף אן אופן וואס פילט זיך כמעט נאטירליך.

מולטימאדאל RAG מיט LLaVA און LangChain (18:27:10)

בילד-באשרייבונג (image captioning) מיט LLaVA (18:27:10)

דער קורס הייבט אן מיט דעם וואס LVLM מאדעלן קענען טון: פארשטיין א קאמפליצירטע טשארט, דערקענען און לייענען טעקסט אין פאטאגראפיעס, און אפכאפן פיינע פרטים אין בילדער מיט הויכע רעזאלוציע. דערנאך ווערן געוויזן עטליכע פראקטישע ביישפילן פון chat completion מיט LLaVA.

דער ערשטער ביישפיל איז image captioning, דאס הייסט אז דער מאדעל זאל אליין שרייבן א באשרייבונג פון א בילד. אזוי איז געבויט דער קאוד:

- מען שרייבט א פראמפט: "please describe the image in detail", אז דער מאדעל זאל באשרייבן דאס בילד אין פרטים.
- מען גיט אן דעם image path, וואו דאס בילד ליגט.
- דער קאוד גרייט צו דעם "שמועס" (conversation), און רופט דעם completion endpoint פון Prediction Guard.
- כדי צו ווייזן דעם רעזולטאט שרייבט מען display(Image.open(image_path)) און דערנאך print(caption).

וועז מען לאוט דאס לויפן, קומט ארויס דאס בילד מיטן קעפשאן: "דאס בילד ווייזט א גרופע פון פינף מענער אין בלויע העמדער וואס שטייען צוזאמען אין א space shuttle, און זיי זענען אויס אין א באקוועמע, אפגעשפאנטע סביבה." אויפן בילד זעט מען טאקע די פינף מענער אין בלויע העמדער. אזוי טוט מען chat completion פאר image captioning: מען גיט נאר דאס בילד, און LLaVA שרייבט א באשרייבונג.

פרעגן וועגן וויזועלע סימנים אין א בילד (18:28:42)

דער צווייטער ביישפיל: מען פרעגט דעם מאדעל א פראגע לויט די וויזועלע סימנים (visual cues) וואס זענען אין בילד. דער פראמפט איז: "What is likely going to happen next" – וואס וועט ווארשיינליך פאסירן נאכדעם, לויט דעם בילד. דער קורס באטאנט אז מען קען שרייבן וועלכן פראמפט מען וויל און פרעגן וואס מען וויל.

ווידער ווייזט מען דאס בילד מיט display(Image.open(path)) און דעם ענטפער מיט print(answer). אויפן בילד שטייט א מאן אויף די הענט (handstand) אויף א skateboard ביים ים-ברעג. דער מאדעל ענטפערט: "אויפן בילד טוט א מאן א handstand אויף א skateboard ביים beach. ס'איז ווארשיינליך אז ער וועט פארלירן די באלאנס און אראפפאלן פון skateboard, ווייל א handstand איז שווער און פאדערט אסאך...". דער מאדעל זאגט אלזא פאראויס, לויט וואס ער זעט אויפן בילד, אז דער מאן וועט אראפפאלן.

דער קורס פאסט צוזאם: אזוי קען מען פרעגן וואס מען וויל בײַ LVLM מאדעלן. דא האט מען געגנוצט LLaVA, אבער מען קען נוצן יעדן אנדערן מאדעל, און מען קען אויך פרעגן נאכפאלגנדע פראגעס (follow-up questions). אין דער קומענדיגער לעקציע ווערט געוויזן מולטימאדאל RAG מיט מולטימאדאל LangChain.

וואס איז מולטימאדאל LangChain (18:30:47)

אין דעם מאדול טוט מען מולטימאדאל RAG מיט מולטימאדאל LangChain, און מען גיט דעם נאטבוק דעם נאמען "multimodal rag with multimodal langchain".

די דעפיניציע: מולטימאדאל LangChain פארברייטערט דאס וואס LangChain קען טון ווייטער פון טעקסט. ס'ברענגט אריין אין דעם AI ארבעטס-פראצעס (workflow) נאך סארטן אינפארמאציע (modalities): בילדער, אודיא, ווידעא, און סטרוקטורירטע דאטא. דאס דערמעגלעכט RAG (retrieval augmented generation), זוכן און inference מיט טעקסט, זעאונג (vision), אודיא און מער.

די הויפט-טיילן פון מולטימאדאל LangChain:

- מולטימאדאל embeddings – זיי פארוואנדלען טעקסט, בילדער, ווידעא און אודיא אין וועקטארן (vector representation). מען נוצט דערפאר מאדעלן ווי) CLIP, BLIP, BridgeTowerוואס זענען שוין פריער ערקלערט געווארן אלס LVLM מאדעלן), אדער OpenAI vision transformer.
- וועקטאר-דאטאבאזעס פאר מולטימאדאל retrieval.
- מולטימאדאל pipelines.
- צוזאמשטעלן LLM מיט LVLM.

ביישפילן פון נוצן: זוכן אין בילדער און ווידעאס, מולטימאדאל chatbots, באארבעטן דאקומענטן, און אפילו אין געזונט-ווען (healthcare) און זיכערהייט.

פארוואס נוצן מולטימאדאל LangChain? עס פארבעסערט RAG אפליקאציעס און RAG pipelines, עס לאזט דעם AI טראכטן ווייטער פון נאר טעקסט, און עס פארבעסערט זוכן און רעקאמענדאציעס.

צוגרייטונג: די imports (18:32:23)

ערשט שרייבט מען די imports:

- import lancedb
- פון utils: load_json_file
- פון bridgetower embeddings: BridgeTowerEmbeddings
- פון mm_rag.vectorstores.multimodal_lancedb: MultimodalLanceDB
- פון mm_rag.MLM.client: PredictionGuardClient
- פון mm_rag.MLM.lvlm: LVLM
- from PIL import Image
- פון langchain_core.runnables: RunnableParallel, RunnablePassthrough, RunnableLambda

דער קורס ערקלערט יעדן פון זיי:

- LanceDB איז א וועקטאר-דאטאבאזע וואס היט אפ און זוכט ארויס מולטימאדאל (embeddingsטעקסט, בילד, ווידעא), און שטיצט שנעלע ענליכקייט-זוכונג (similarity search) פאר RAG-באזירטע AI מאדעלן.
- utils ברענגט אריין הילפס-פונקציעס; load_json_file לאדט ארויף metadata און embeddings וואס זענען אפגעהיטן אין JSON פארמאט.
- BridgeTowerEmbeddings איז א vision-language מאדעל (LVLM) וואס פארוואנדלט טעקסט און בילדער אין וועקטאר-embeddings פאר מולטימאדאל retrieval.
- MultimodalLanceDB איז א געמאכטער "ראפער" אויף LanceDB צו באהאנדלען מולטימאדאל דאטא (טעקסט מיט בילדער), און ערלויבט אפהיטן און ארויסנעמען embeddings.
- PredictionGuardClient גיט AI-באזירטע inference פאר מולטימאדאל RAG.
- LVLM איז א מאדעל וואס באארבעט פראגעס מיט בילד און טעקסט.
- PIL Image באהאנדלט לאדן און צוגרייטן (pre-processing) פון בילדער.
- RunnableParallel לאזט לויפן עטליכע טיילן אין דער זעלבער צייט, ווי ארויסנעמען בילד און טעקסט.
- RunnablePassthrough גיט ווייטער דאטא אן קיין ענדערונג.
- RunnableLambda וועגדט אן א געמאכטע טראנספארמאציע אויף דער דאטא.

איינשטעלן LanceDB און דעם retriever (18:37:18)

צו קאנפיגורירן LanceDB פאר מולטימאדאל retrieval דעקלארירט מען ערשט דעם host file: LANCEDB_HOST_FILE = "./shared_data/.lancedb/", און דערנאך דעם נאמען פון דער טאבעלע (TBL_NAME, מיט א נאמען וואס ענדיגט זיך מיט "test"). אזוי שטעלט מען איין דעם Lance וועקטאר-סטאר.

דערנאך איניציאליזירט מען דעם embedding מאדעל: embedder = BridgeTowerEmbeddings().

כדי צו מאכן א retriever:

- מען שאפט א LanceDB וועקטאר-סטאר (MultimodalLanceDB).
- מען מאכט פון אים א retriever מיט search_type="similarity" און search_kwargs מיט k=1, דאס הייסט מען וויל נאר דעם איינעם בעסטן רעזולטאט.
- מען רופט אן (invoke) דעם retriever מיט דער פראגע פון באניצער.
- דער קאוד נעמט דעם ערשטן ווידעא-שטיק (video segment) וואס איז ארויסגעקומען, און ציט ארויס פון אים די metadata: דאס בילד (frame) וואס איז ארויסגענומען גענוארן, דעם טראנסקריפט, דעם path פונעם ווידעא פון וועלכן דער frame איז ארויסגענומען, און דעם timestamp ווען דער frame איז ארויסגענומען גענוארן.
- צום סוף ווייזט מען אלעס אויפן עקראן.

דער רעזולטאט: ס'קומט סיי א טראנסקריפט סיי א בילד. די פראגע איז געווען "What do the astronauts feel about their work" – וואס פילן די אסטראנאוטן וועגן זייער ארבעט. דער מאדעל האט געטראפן א שטיק וואו א געוויסער מענטש רעדט, און דער טראנסקריפט זאגט: "וועז איך קוק צוריק אויף דער מיסיע וואס מיר האבן געהאט דא אויף דער International Space Station, בין איך שטאלץ אז איך בין געווען א טייל פון

אסאך פון די וויסנשאפטלעכע אקטיוויטעטן." דערנאך ווייזט זיך דער path פונעם frame, דער path פונעם וויידעא און דער timestamp.

דער LVLM inference מאדול (18:40:53)

דער נעקסטער שריט איז דער LVLM inference מאדול. מען איניציאליזירט:

PredictionGuardClient() = client

LVLM(client=client) = lvlm

דערנאך רופט מען אן דעם LVLM מיט דער פראגע פון באניצער. מען בויט אן "augmented query " מיט א template: "The transcript associated with the image is" – און דערנאך קומט דער טראנסקריפט וואס איז ארויסגעקומען, און נאכדעם די פריערדיגע פראגע. וועו מען דרוקט דאס אויס, זעט מען דעם צוזאמגעשטעלטן פראמפט: דער טראנסקריפט פון בילד ("As I look back on the mission…") מיט דער פראגע.

דערנאך נוצט מען דעם augmented query צוזאמען מיטן path פונעם ארויסגענומענעם בילד אלס אינפוט פאר דעם LVLM. דער LVLM ענטפערט: "די אסטראנאווטן אויפן בילד זעען אויס שטאלץ מיט זייער ארבעט אויף דער International Space Station. זיי טראגן אלע העמדער און שלייפעס, וואס ווייזט אויף א פראפעסיאנעלער...". דער קורס באמערקט אז א טייל דערפון איז נישט ריכטיג – ווארשיינליך האט דער מאדעל זיך פארקוקט אויף די העמדער.

דער פראמפט-באארבעטונג מאדול (18:43:24)

יעצט מאכט מען א פראמפט-באארבעטונג מאדול. מען דעפינירט א פונקציע (prompt processing) וואס נעמט די פראגע פון באניצער מיט דער ארויסגענומענער ווידעא-metadata, פארמאטירט דעם אינפוט, און גרייט אים צו פאר דעם LVLM, דורך LangChain RunnableLambda. די שריט אין דער פונקציע:

- זי באקומט די ארויסגענומענע רעזולטאטן (retrieved results) – ווידעא-frames מיט טראנסקריפטן פונעם וועקטאר-סטאר – און די אריגינעלע פראגע פון באניצער.
- פון די retrieved results קלייבט מען אויס דעם רעלעוואנטסטן, דאס הייסט דעם בעסטן ענליכקייט-מאטש (top ranked).
- מען דעפינירט דעם prompt template מיט דער אריגינעלער פראגע.
- מען ציט ארויס די metadata פונעם אויסגעקלויבענעם רעזולטאט.
- מען פארמאטירט דעם באארבעטן אויסגאנג.
- צום סוף איניציאליזירט מען א LangChain RunnableLambda מיט דער פונקציע.

דערנאך רופט מען אן דעם פראמפט-מאדול מיט די retrieved results און דער פראגע פון באניצער, און דערנאך גיט מען דעם אינפוט צום LVLM. אינעם רעזולטאט זעט מען דעם פראמפט: "The transcript associated with the image is…", מיט דעם טראנסקריפט און דער פראגע.

צוזאמשטעלן די גאנצע מולטימאדאל RAG קייט (18:45:30)

צום סוף דעפינירט מען די מולטימאדאל RAG סיסטעם אלס א chain אין LangChain: מען פארבינדט אלע מאדולן אין איין קייט. מען רופט אן די RAG סיסטעם מיט א פראגע, פונקט ווי פריער – "What do the astronauts feel about their work" – און ס'קומט א ענטפער.

דערנאך פרובירט מען א צווייטע פראגע: "What is the name of one of the astronauts" – וואס איז דער נאמען פון איינעם פון די אסטראנאוטן. דער ענטפער: Robert Behnken. די מולטימאדאל RAG ענטפערט ריכטיג.

אבער דא קומט נאר טעקסט, און מען וויל אויך דאס בילד. דערפאר מאכט מען א מולטימאדאל RAG chain "with retrieved image", וואס גיט צוריק דעם ענטפער צוזאמען מיטן בילד. מען לאזט לויפן די זעלבע צווייטע פראגע, און יעצט קומט ארויס סיי דאס בילד סיי דער path פונעם ארויסגענומענעם frame. דערנאך ציט מען ארויס דעם לעצטן טעקסט-ענטפער צוזאמען מיטן path פונעם frame, און דער נאמען Robert Behnken קומט ווידער ארויס.

דער חילוק: אין דער ערשטער קייט קומט נאר טעקסט, און אין דער צווייטער קומט אויך דער frame פונעם פאסיגן בילד. דער קורס פאסט צוזאם: אזוי קען מען פרעגן וואס מען וויל בײַ א מולטימאדאל RAG מיט מולטימאדאל LangChain, און מען הייבט אן פרובירן נאך א פראגע.

וועב-קראולינג, RAG און Crawl4AI (18:47:42)

סוף פון דעם מולטימאדאל RAG ביישפיל (18:47:42)

דער ביישפיל פון מולטימאדאל RAG מיט LangChain ווערט פארענדיקט. די פראגע וואס מען גיט דעם מאדעל איז: באשרייב א בילד פון אן אסטראנאוט וואס מאכט א "spacewalk" (א גאנג אין דרויסנדיקן רוים), מיט א וואונדערליכער אויסזיכט אויף דער ערד פון רוים הינטער אים. דער רעזולטאט ווייזט טאקע א spacewalk, נאר די ערד זעט מען נישט.

דעריבער ענדערט מען א ביסל די פראגע און מען שרייבט נאך א נייע פראגע, "query six". מען לויפט עס ווידער, און יעצט קומט ארויס סיי א ריכטיגער spacewalk, סיי די ערד. אזוי, ווערט ערקלערט, ניצט מען מולטימאדאל RAG מיט מולטימאדאל LangChain.

וואס איז וועב-קראולינג (18:48:43)

דער קורס הייבט אן א נייעם טעמע מיט א משל: שטעלט אייך פאר אז איר זענט א רעפארטער וואס באריכטעט וועגן א גרויסער פאסירונג. אנשטאט צו גיין אליין צו יעדן טייל פון דער פראגראם און רעדן מיט יעדן גאסט, שיקט איר א ראבאט וואס זאמלט פאר אייך אלע נייעס. דער ראבאט באזוכט יעדן ווינקל פון דער פאסירונג, שרייבט אויף אלע וויכטיגע פרטים און ברענגט זיי צוריק, אז איר זאלט קענען שרייבן דעם באריכט. דער ראבאט איז א "web crawler".

אויף פשוט'ע ווערטער: וועב-קראולינג איז א פראצעס אין וועלכן א קאמפיוטער-פראגראם אדער ראבאט באזוכט אויטאמאטיש וועבזייטלעך, נעמט ארויס די דאטא און ברענגט זי צוריק אין א גוט-סדר'דיגער פארעם. עס שפארט א סך צייט, ווייל עס זאמלט אסאך אינפארמאציע פון אסאך זייטלעך אויף איין מאל.

ביישפיל: איר ווילט נאכפאלגן די לעצטע פרייזן פון פראדוקטן אויף פארשידענע e-commerce זייטלעך. אנשטאט צו באזוכן יעדן זייטל מיט דער האנט און קוקן אויף די פרייזן, ניצט מען א web crawler וואס באזוכט אויטאמאטיש די זייטלעך און נעמט ארויס די פרייזן.

פארוואס איז וועב-קראולינג וויכטיג פאר RAG? וועו מען שפייזט דעם AI מיט וויסן, העלפן די web crawlers צוזאמנעמען דאטא אויף אן אופן וואס מאכט זי נוצלעך פארן AI. וואס מער דאטא דער AI קען דערגרייכן, אלס קלוגער וועט ער.

וואס איז RAG (18:49:45)

RAG מיינט Retrieval-Augmented Generation. עס קען אויסזען קאמפליצירט, אבער מען ערקלערט עס מיט א משל: שטעלט אייך פאר אן AI-געהילף. געווענליך ווייסט ער אסאך וועגן אלגעמיינע טעמעס, אבער וועו מען פרעגט אים וועגן עפעס נייעס אדער ספעציפיש, טרעפט ער בלויז. עס איז ווי מען וואלט געפרעגט איינעם וואס ווייסט אסאך אלגעמיינע פאקטן, אבער האט נישט געליינט דאס לעצטע בוך אויף אייער באליבטן טעמע.

RAG לייזט דעם פראבלעם: דער AI קען ארויסנעמען אינפארמאציע פון דרויסנדיקע מקורות, ווי וועבזייטלעך און דאקומענטן, און דערנאך שאפן אן ענטפער אויפן יסוד פון יענער אינפארמאציע. עס איז ווי מען גיט דעם געהילף א "סופער-כח" צו זוכן ענטפערס אליין, אנשטאט זיך צו פארלאזן נאר אויף דעם וואס ער ווייסט שוין. דער בעסטער טיייל: עס העלפט דעם AI ווערן א מומחה אין ספעציפישע געביטן וואס ער האט פריער נישט געקענט.

ביישפיל: מען בויט א "customer support bot" פאר א נייעם פראדוקט. געווענליך וואלט דער באט נישט געווואוסט אסאך וועגן דעם פראדוקט, ווייל ער איז ניי אדער נישט'דיג. אבער מיט RAG קען מען אים שפייזן מיט דער גאנצער פראדוקט-דאקומענטאציע און פאקטן. דאן וועט דער באט קענען געבן קענען געניע און דעטאלירטע ענטפערס, ווי א מומחה וואס איז טרייניירט געווארן אויף אלעם וואס שייך צו יענעם פראדוקט.

ווי וועב-קראולינג ווערט געניצט אין RAG (18:51:17)

דער קשר איז אזוי: וועב-קראולינג איז דער ערשטער שריט פון RAG. אויב מען וויל אז דער AI זאל ווערן א מומחה אין א געוויסן ענין, דארף ער האבן צוטריט צו דעטאלירטער אינפארמאציע, און דא קומען אריין די web crawlers. זיי העלפן צוזאמנעמען די דאטא וואס דער AI וועט ניצן צו שאפן קלוגע ענטפערס.

כדי RAG זאל ארבעטן, מוז מען שפייזן דעם AI מיט דער ריכטיגער אינפארמאציע. מען קען ניצן web crawlers צו באזוכן פארשידענע וועבזייטלעך און צוזאמנעמען די געוואונטשענע אינפארמאציע, למשל:

- פראדוקט-פרטים
- באניצער-אנווייזונגען (user manuals)
- טעכנישע בלאגס

דערנאך האלט מען אן די אינפארמאציע אין א פארמאט וואס דער AI קען פארשטיין.

ביישפיל: מען בויט א טשעט-באט וואס העלפט מענטשן פארשטיין א געוויסע פראגראמירונג-שפראך. מען ניצט א web crawler צו "scrape"ן אלע tutorials, גיידס און דאקומענטאציע וועגן יענער שפראך וואס געפינען זיך אנליין, און מען שפייזט עס אריין אין דעם AI. וועו באניצער פרעגן דעם באט פראגעס, קען ער ענטפערן מיט דער מערסט-לעצטיגער און שייכותדיגער אינפארמאציע.

וואס איז Crawl4AI (18:52:20)

דער ווייטערדיגער טעמע איז Crawl4AI און ווי עס ארבעט. מען גייט אריין אויף זיין וועבזייטל; עס איז open source און מען קען עס גלייך ניצן. לויט דער באשרייבונג איז Crawl4AI דער נומער-איינס "trending GitHub repository", וואס ווערט אקטיוו אויפגעהאלטן דורך א לעבעדיגער קהילה. עס גיט שנעלע, "AI-ready" קראולינג, צוגעפאסט פאר large language models, AI-אגענטן און אויך דאטא-pipelines . עס איז open source, פלעקסיבל און געבויט פאר "realtime" פערפארמענס, און גיט פראגראמירער אומגעהויערע שנעלקייט, פרעציזיע און גרינגקייט צו "deploy"ן. אויפן זייטל זעט מען אויך דעם לעצטן "update".

די הויפט-נייעס: א "deep crawling " סיסטעם, וואס לאזט אויספארשן וועבזייטלעך ווייטער פון די ערשטע URLs, מיט דריי סטראטעגיעס: BFS, DFS און best-first, פלוס "page limiting". מען באקומט אויך די מעגליכקייט פון עטלעכע קראולינג-סטראטעגיעס.

פארוואס ניצן Crawl4AI (18:53:24)

דער קורס רעכנט אויס די סיבות:

- געבויט פאר large language models: עס שאפט קלוגע, קורצע markdown, אפטימירט פאר RAG און fine-tuning.
- שנעל: רעזולטאטן זעקס מאל שנעלער, מיט realtime און ביליגער פערפארמענס.
- פלעקסיבלע בראוזער-קאנטראל: session management, proxies און custom hooks פאר גלאטן צוטריט צו דאטא.
- "heuristic intelligence": אוואנסירטע אלגאריטמען פאר עפעקטיוון ארויסנעמען, וואס פארמינערט די אפהענגיקייט אויף טייערע מאדעלן.
- אינגאנצן open source, אן API keys, גרייט פאר Docker און cloud.

דערנאך ווייזט מען מיט א ביישפיל פארוואס עס איז אזוי וויכטיג. מען גייט אויף א וועבזייטל און קליקט "view page source", און זעט דעם רויען HTML. עס זעט אויס ווי א פולשטענדיגער בלאָטע; אפילו פאר א מענטש איז שווער דארט צו געפינען די נוצלעכע אינפארמאציע, און פאר א large language model איז עס מסתמא נאך שווערער. איינס פון די וויכטיגסטע זאכן וואס Crawl4AI טוט: עס נעמט דעם מיאוסן HTML און פארוואנדלט אים אין markdown. דאס איז א ליינבארער פארמאט אויך פאר מענטשן, א זייער ריינער אופן צו ווייזן א וועבזייטל מיט אלע אינפארמאציע וואס מען זעט אויפן UI. עס ווערט אלעס טעקסט, און מען קען עס געבן א large language model, אז ער זאל עס בעסער פארשטיין.

נאך א פונקט: Crawl4AI באהאנדלט הינטער די קוליסן אסאך זאכן וואס זענען נישט גרינג, ווי proxies און session management. דעריבער קען מען נישט אזוי גרינג מאכן אן אייגענע ווערסיע נאר ווייל מען ווייסט ווי צו ניצן דעם requests מאדול אין Python צו ציען HTML. עס איז אינגאנצן open source און גרינג צו deploy'ן, אויך דורך Docker; אבער דער קורס קאנצענטרירט זיך יעצט נישט אויף Docker.

אינסטאלאציע און באשטעטיגונג (18:55:25)

די שריט:

- מען קאפירט דעם אינסטאלאציע-באפעל פונעם זייטל, עפנט א terminal און פעיסט עס אריין. עס קומט א ווארנונג אז דער טעקסט האט עטלעכע שורות; מען קליקט "paste anyway". אין דער לעקציע איז

Crawl4AI שוין געווען אינסטאלירט, און די אנדערע באפעלן וועלן אויטאמאטיש אויסגעפירט, אריינגערעכנט די pre-release וועלסיעס.

- אין עטלעכע סעקונדעס וועלט Crawl4AI הצלחה'דיג אינסטאלירט, און דערנאך די pre-release וועלסיעס.
- מען לויפט דעם post-installation setup מיט דעם באפעל וואס שטייט אויפן זייטל.
- צו באשטעטיגן די אינסטאלאציע גיט מען דעם באפעל crawl4ai-doctor. עס לויפט א "health check" וואס טעסט אלע קראולינג-פעאיקייטן. אלע טעסטן זענען דורכגעגאנגען.
- אויב עס זענען דא פראבלעמען מיט דעם בראוזער, קען מען אים אינסטאלירן מיט דער האנט מיט א באפעל פונעם זייטל.

דערנאך קען מען לויפן א פשוטן web crawl סיי מיט Python, סיי מיט דער נייער command-line interface. מען דערמאנט אויך אז מען דארף אינסטאלירן דעם Visual Studio installer, כדי צו האבן די נויטיגע C++ כלים און CLI.

קראולן איין זייטל: PydanticAI (18:57:00)

אין VS Code ווייזט מען דעם קאוד, די טעקע crawl_single_page.py, וואס מאכט א פשוטן web crawl מיט Python פאר דעם URL פון PydanticAI. מען ערקלערט וואס דאס איז: PydanticAI איז א Python אגענט-framework, געמאכט צו פארגרינגערן דאס בויען פראדוקציע-אפליקאציעס מיט generative AI. פארוואס ניצן עס: עס איז געבויט דורכן Pydantic-טים, עס איז "model agnostic", האט Logfire אינטעגראציע, "type safe", א Python-צענטריישן דיזיין, structured responses, א dependency injection סיסטעם, stream responses, graph support און נאך.

מען ווייזט ווידער "view page source" פון דעם זייטל, דעם HTML. דערנאך גייט מען צום terminal און לויפט python מיטן נאמען פון סקריפט (1-crawl_single_page.py). אין א סעקונדע קומט ארויס דער גאנצער אינהאלט און סטרוקטור פונעם זייטל אין א זייער ליינבארן פארמאט: די הקדמה, וואס איז דער framework, און די קאוד-ביישפילן.

sitemap פאר אלע זייטלעך אויף איין מאל (18:59:01)

אין דער navigation bar זעט מען אסאך זייטלעך, און יעדן באזונדער וואלט מען געדארפט באזונדער קאודירן. כדי צו נעמען אלע זייטלעך אויף איין מאל, ניצט מען א sitemap. א sitemap גיט די גאנצע סטרוקטור פונעם וועבזייטל, מיט אלע בלעטער, און דערנאך קען מען פון דארט scrape'ן די דאטא. מען שרייבט צום סוף פון דער אדרעס פון יעדן זייטל: /sitemap.xml.

ביי PydanticAI קומט ארויס אן XML טעקע, מיט א הערה אז עס איז נישט פארבונדן קיין "style" אינפארמאציע, און דער דאקומענט וועלט געוויזן. דאס איז די גאנצע סטרוקטור פון אלע זייטלעך פון דער PydanticAI דאקומענטאציע.

דאסזעלבע טוט מען פאר דער Crawl4AI דאקומענטאציע: מען גייט אויף איר זייטל, צוגיט /sitemap.xml און זעט די sitemap מיט אלע זייטלעך, פון וואנעט מען קען scrape'ן די דאטא. אזוי קען מען טון ביי רוב וועבזייטלעך, בעיקר e-commerce געשעפטן געבויט מיט Shopify אדער WordPress, ווייל זיי האבן אסאך זייטלעך. אויך געווענליכע וועבזייטלעך האבן עס, צוליב search engine optimization.

robots.txt און sitemap.xml – עטיק ביים סקרעיפן (19:01:05)

דער קורס ערקלערט אז אסאך וועבזייטלעך ווילן גאר אז מען זאל זיי "קראולן", ווייל זיי ווילן אז זייער אינפארמאציע זאל זיין בריט פארשפרייט. למשל, וועך עס בויט אן AI-פרעימווארק וויל אז מענטשן זאלן דאס קראולן און בויען אגענטן ארום דעם, ווייל דעמאלט באניצט מען זיין פרעימווארק, און עס וועלן מער צוגענגליך.

פונדעסטוועגן ליגט אסאך עטיק אינעם סקרעיפן פון א וועבזייטל. כדי צו וויסן וואס מען טאר, קען מען גיין צו יעדן געווענליכן וועבזייטל, למשל youtube.com, און צולייגן אין סוף /robots.txt . מען זעט דעמאלט דעם robots.txt פייל פון YouTube. אין אים שטייט א שפאסיגע הקדמה, אז ער איז "געשאפן געווארן אין דער ווייטער צוקונפט, אין יאר 2000, נאך דעם ראבאטישן אויפשטאנד פון מיטן פון די 90ער יארן וואס האט אויסגעמעקט אלע מענטשן". דערנאך קומען די שורות user-agent און disallow, וואס באשטימען וועלכע טיילן מען טאר יא און וועלכע נישט.

דער עצה איז: פאר יעדן וועבזייטל וואס מען וויל סקרעיפן, זאל מען צוערשט קוקן אין robots.txt און ערשט דערנאך גיין ווייטער. GitHub איז אויך א גוטע דוגמא. מען שרייבט github.com/robots.txt, און דארט זעט מען אלע ערלויבענישן: ווי אזוי מען טאר סקרעיפן און וועלכע עטישע כללים מען דארף האלטן. עס שטייט אז וועך עס וויל קראולן GitHub דארף זיך פארבינדן מיט זייער סופארט-גרופע. מען דערמאנט אויך אז GitHub שטעלט צו א בריטע API, און עס שטייען אויך דער user-agent נאמען, דער crawl-delay)וויפיל מען דארף ווארטן צווישן איין אנפרעג און דער צווייטער) און אנדערע פרטים.

עס זענען פאראן צוויי אדרעסן וואס מען קען צולייגן כמעט צו יעדן וועבזייטל: /robots.txt אדער / sitemap.xml. דער פלאן אינעם קאוד איז צו נעמען דעם sitemap, ארויסציען פון אים יעדן איינציגן URL, און זיי געבן פאר Crawl4AI. מען וויל דאס טון אויף אן עפעקטיווע אופן, ווייל דער ערשטער שריט איז בלויז צו נעמען אלע URL'ס, דורכגיין זיי אין א לופ און באארבעטן איינס נאכן אנדערן.

סעקווענציעלע קראולינג מיט דעם sitemap (19:03:08)

אין VS Code זעט מען ווייטער: פריער האט מען באקומען די אינפארמאציע וועגן די סטרוקטור פון איין בלאט פונעם וועבזייטל, און דאס זעלבע קען מען טון פאר אנדערע בלעטער. דער קומענדיגער סקריפט הייסט crawl_docs_sequential.py. דאס איז א פשוטער קאוד צו קראולן דעם וועבזייטל:

- מען שאפט דעם crawler מיט דער פונקציע וואס איז דא דערפאר.
- מען נעמט דעם URL פון די Pydantic AI דאקומענטאציע.
- מען גיט אריין דעם sitemap.xml כדי צו באקומען די URL'ס. דער sitemap URL שטייט אינעם קאוד.
- מען ציט ארויס אלע URL'ס פונעם sitemap. דער namespace וועלט גענענליך דעפינירט אינעם root element פון דעם XML.

דערנאך גייט מען אריין אינעם טערמינאל, לאזט לויפן דעם זעלבן באפעל ווי פריער, און שרייבט דעם נאמען פונעם צווייטן סקריפט. דער רעזולטאט: עס וועלט געוויזן אז מען האט געפונען 63 URL'ס צו קראולן, מיט "sequential crawling with session reuse", דאס הייסט סעקווענציעל קראולן און דערביי ווידער באניצן די זעלבע סעסיע. דער וועבזייטל וועלט מיט הצלחה געקראולט.

עס קומט אויך ארויס א פעלער-מעלדונג: דער markdown_v2 אטריבוט איז "deprecated" (ער ווערט מער נישט רעקאמענדירט) און מען האט אים אראפגענומען. מען דארף נוצן markdown , וואס גיט יעצט צוריק א markdown generation result מיט געוויסע אייגנשאפטן. אזוי באקומט מען די פרטים פון דעם קראול פאר יענעם וועבזייטל, און מען זעט וואס מען דארף ענדערן כדי עס זאל ארבעטן.

פארגעשריטענע טעכניקן: BFS, DFS, Best-First און dispatchers (19:04:42)

דער קורס גייט איבער צו פארגעשריטענע קראולינג-טעכניקן מיט Crawl4AI. מען קען קראולן אסאך בלעטער אויף אמאל (פאראלעל), וואס מאכט די ארבעט אסאך גרינגער. אין דער דאקומענטאציע ווייזט מען דעם טייל "multi-URL crawling". אנשטאט באזוכן איין בלאט אויף אמאל, לאזט Crawl4AI אויספארשן א וועבזייטל אויף פארשידענע אופנים. די סטראטעגיעס העלפן קראולן עפעקטיוו און זיכער מאכן אז מען באקומט די מערסט רעלעוואנטע בלעטער:

- Breadth-First Search (BFS): פארשט אויס דעם וועבזייטל שטאפל נאך שטאפל. מען סקרעיפט אלע בלעטער אויף דער זעלבער מדרגה איידער מען גייט צום קומענדיגן שטאפל.
- Depth-First Search (DFS): טויכט טיף אריין אין דעם וועבזייטל און סקרעיפט יעדן אפטייל אינגאנצן איידער מען גייט איבער צום קומענדיגן צווייג.
- Best-First: נוצט א מעסטונג (scoring) כדי צו געבן פריאריטעט פאר די וויכטיגסטע בלעטער ערשט.

ווען מען קראולט אסאך URL'ס, איז דער בסיסדיגער וועג צו נוצן arun אין א לופ. דאס איז פשוט אבער ווייניגער עפעקטיוו. אויב מען נוצט אן אנדערע פונקציע (פאר אסאך URL'ס צוגלייך), באהאנדלט זי עפעקטיוו אסאך URL'ס מיט ריכטיגע קאנטראל איבער צוגלייכצייטיגקייט (concurrency). פאר די בעסטע רעזולטאטן קען מען צופאסן ווי דער dispatcher אויפפירט זיך, לויט די באדערפענישן: באהאנדלונג פון זכרון, rate limits, א.א.וו.

פארוואס נוצט מען ?dispatchers צוליב דריי זאכן: אדאפטיוועט באגרענעצונג פונעם טעמפּא (adaptive rate limiting), מאניטארינג אין רעאלע צייט, און פלעקסיביליטעט.

RateLimiter און CrawlerMonitor (19:05:42)

די הויפט-קאמפאנענטן זענען צוערשט דער RateLimiter. דאס איז א געצייג וואס העלפט קאנטראלירן דעם טעמפּא פון די אנפרעגן, כדי מען זאל נישט איבערלאדן די סערווערס אדער ווערן בלאקירט צוליב די rate limits. ער ארבעט אינעווייניג: ער פארהאלט די אנפרעגן און באהאנדלט ווידערפרואוון (retries), אבער מען קען אים איינשטעלן דורך די פאראמעטערס פון זיין constructor . איינער פון די פאראמעטערס איז base_delay. אין דער דוגמא קען דער פארהאלט ווערן אויסגעקליבן צופעליג צווישן 2.3 סעקונדעס און 4.1 סעקונדעס. אין דער דאקומענטאציע איז אויך פאראן א דוגמא ווי אזוי צו אינישאליזירן און נוצן דעם RateLimiter אין א פראיעקט.

די צווייטע קאמפאנענט איז דער CrawlerMonitor. ער גיט א בליק אין רעאלע צייט אויף וואס עס טוט זיך ביים קראולן. די סטראטעגיעס זאגן צו אז Crawl4AI זאל ארויסציען די וויכטיגסטע דאטא פאר דעם AI, שנעל און עפעקטיוו.

דער קורס פארגלייכט: ווען מען האט געלאזט לויפן crawl_docs_sequential.py, האט מען געמוזט קראולן אלע בלעטער איינס נאכן אנדערן. מיט פאראלעלער פראצעסירונג קען מען דאס מאכן אסאך שנעלער, אפילו אין א פאר סעקונדעס.

פאראלעלער קראול און דער Pydantic AI אגענט (19:06:45)

דער דריטער קאוד הייסט crawl_docs_parallel.py. דא קען מען אויך צולייגן דעם עלטערן-פאלדער (parent directory) צום סיסטעם-פאט דורך א באפעל דערפאר. מען שאפט דעם crawler מיט דער פונקציע, און דערנאך טיילט מען די URL'ס אין גרופעס (batches), יעדע גרופע אזוי גרויס ווי max_concurrent, די מאקסימאלע צאל וואס לויפט צוגלייך. דער קאוד ציט ארויס אלע URL 's פון דער Pydantic AI דאקומענטאציע, און מען נוצט דעם sitemap כדי זיי צו באקומען.

דערנאך ווייזט מען ווי אזוי מען פארבינדט דאס מיט אן AI אגענט. פון pydantic_ai אימפארטירט מען Agent, און מען נוצט אויך דעם OpenAI מאדעל (OpenAIModel) כדי צו שאפן דעם אגענט און אריינלייגן די גאנצע אינפארמאציע אינעם knowledge base)די וויסן-באזע). מען גיט אים א system prompt, בערך אזוי: "דו ביסט אן עקספערט אין Pydantic AI, א Python אגענט-פרעימווארק. דו האסט צוטריט צו דער גאנצער דאקומענטאציע." ווייטער שטייט אינעם פראמפט:

- וועז מען קוקט ערשט אין דער דאקומענטאציע, זאל מען שטענדיג אנהייבן מיט RAG.
- מען זאל אויך קוקן אין דער ליסטע פון די צוגענגליכע דאקומענטאציע-בלעטער, און ארויסנעמען דעם אינהאלט פון די בלעטער וועז דאס וועט העלפן.
- מען זאל שטענדיג ערליך זאגן דעם באניצער וועז מען האט נישט געפונען דעם ענטפער אין דער דאקומענטאציע אדער אינעם ריכטיגן URL.

דערנאך קומען די געצייג-פונקציעס פונעם אגענט. איינע נעמט ארויס א ליסטע פון אלע צוגענגליכע Pydantic AI דאקומענטאציע-בלעטער און גיט צוריק די ליסטע פון אלע באזונדערע (unique) URL'ס. א צווייטע נעמט ארויס דעם פולן אינהאלט פון א געווישן דאקומענטאציע-בלאט, דורך צוזאמשטעלן אלע זיינע שטיקלעך (chunks). דער קורס זאגט אז דאס איז גאנץ פשוט, און אזוי קען מען בויען אן אייגענעם chatbot אגענט.

פארוואס Crawl4AI פאסט פאר RAG (19:08:47)

צום סוף פון דעם טייל ווערט ערקלערט פארוואס Crawl4AI פאסט אזוי גוט פאר RAG. מען קען דערמיט סקרעיפן וועבזייטלעך און זיי פארוואנדלען אין דאטא-קוואלן פון הויכער קוואליטעט פארן AI:

- גיכקייט: עס סקרעיפט דאטא אסאך שנעלער ווי אנדערע געצייג.
- שפארעוודיג מיטן זכרון: עס איז געמאכט צו באהאנדלען גרויסע מאסן דאטא אן נוצן צופיל זכרון. מען קען אזוי סקרעיפן גאנצע וועבזייטלעך אן דאס דער סיסטעם זאל "קראשן".
- פלעקסיבלע קראולינג: מען קען קראולן א פאר בלעטער אדער טויזנטער, לויט די באדערפענישן.

די צוואמשטעל פון גיכקייט און עפעקטיווקייט מאכט Crawl4AI א פיינעם געצייג פאר RAG. עס לאזט דעם AI באקומען דאס מערסט דערהיינטיגטע וויסן אין רעקארד-צייט. די הויפט-נקודה: דורך זאמלען רעלעוואנטע דאטא, קראולן דאס וועב, אויסרייניגן די דאטא און זי אריינגעבן אין דעם AI מאדעל, קען מען מאכן דעם AI אן עקספערט אין סיי וועלכן נושא.

א נייער טייל: פּלאן פון דער AI-אגענטן לעקציע (19:09:49)

דער קורס הייבט אן א נייעם טייל פאר אנהייבער. די אגענדע:

- וואס איז אן AI אגענט.

- די וויכטיגע אונטערשיידן צווישן AI אגענטן און גרויסע שפראך-מאדעלן (LLMs).
- ווי אזוי אן AI אגענט ארבעט.
- א קורצע איבערבליק אויף דער צוקונפט און די ריזיקעס פון AI אגענטן.
- לעצטע געדאנקען.
- א לייוו פראקטישע דעמאנסטראציע ווי אזוי צו בויען AI אגענטן.

וואס איז אן AI אגענט – דער ביישפיל מיט Tony (19:10:20)

אינעם קערן איז אן AI אגענט א ווייכווארג-פראגראם וואס איז געבויט אויסצופירן אויפגאבעס אויטאנאם, אליין. ניט ווי טראדיציאנעלע ווייכווארג וואס פאלגט שטרענג פאראויס-באשטימטע כללים, קענען AI אגענטן מאכן באשלוסן לויט פארשטאנד, אינטעראקציע און דאטא אין רעאלע צייט. זיי נוצן די נייסטע AI מאדעלן ווי GPT, Claude און Gemini כדי צו אנאליזירן, פראצעסירן און אויספירן אויפגאבעס אויף א דינאמישן אופן. מען קען זיך דאס פארשטעלן ווי א דיגיטאלער געהילף וואס פאלגט נישט נאר שטייפע אנווייזונגען.

די דוגמא: א געוועלכער באפעל צו א דיגיטאלן געהילף איז "קוק אויב Tony איז פריי פרייטאג און באשטעל א מיטינג". דאס איז פשוט און גלייך צו פארשטיין. אבער אויב מען ענדערט א פאר וועריעבלס אין דעם באפעל, למשל "באשטעל א מיטינג מיט Tony אויף דער פריסטער באקוועמער צייט אינעם קומענדיגן חודש", זעט עס אויס ענליך, אבער דער AI אגענט מוז דורכגיין עטליכע שריט:

- קוקן אינעם קאלענדאר נאך פרייע צייטן. דער אגענט לאגט זיך אריין אין דעם אקאונט פונעם באקומער, Tony'ס אקאונט, און קוקט אויב עס זענען פאראן פרייע צייטן פאר א מיטינג.
- באשטימען די בעסטע צייט.
- ארויסשיקן די איינלאדונגס.

דאס אלעס פאסירט אויטאנאם. אינעם ערשטן פאל, אנדערש, קוקט דער דיגיטאלער געהילף גלייך אויף פרייטאג אינעם קאלענדאר און באשטעלט דעם מיטינג אויף דער צייט וואס מען האט אים געגעבן, סיי Tony איז פריי און סיי נישט. דאס איז דער אונטערשייד צווישן א דיגיטאלן געהילף און אן AI אגענט.

אונטערשייד צווישן AI אגענטן און LLMs; "האלוצינאציעס" (19:12:27)

כאטש AI אגענטן נוצן גרויסע שפראך-מאדעלן ווי GPT-4, גייען זיי ווייטער ווי בלויז שאפן טעקסט. LLMs שאפן ענטפערס לויט די דאטא אויף וועלכע מען האט זיי טריינירט, אבער זיי האנדלען נישט אקטיוו מיט דער ווירקליכער וועלט. AI אגענטן, פארקערט, קענען צוקומען צו דרויסנדיגע דאטא, פלאנירן אויפגאבעס און אויספירן אקציעס, אלעס אין רעאלע צייט. דאס איז איינער פון די וויכטיגסטע אונטערשיידן.

א ביישפיל: אויב מען פרעגט א LLM וועגן א ספארט-שפיל וואס איז פארגעקומען נעכטן, האט עס מעגליך נישט קיין דערהיינטיגט וויסן. אן AI אגענט, אבער, קען אראפנעמען לייוו דאטא פונעם וועב, פון דאטא-באזעס אדער פון APIs, און געבן א ענוויען ענטפער.

דערנאך וועלט דערמאנט דער באגריף "האלוצינאציעס". דער באגריף קומט פון מענטשן, אבער LLMs קענען אויך "האלוצינירן". למשל, וועז מען פרעגט אן אומרעאליסטישע שאלה: עס זענען פאראן פארשידענע סארטן עפל, ווי Fuji, McIntosh און Black Oxford. אויב מען שרייבט אבער "דערצייל מיר וועגן א וואסערמעלאן-עפל" אדער "א מאראנץ-מאנגא", וואס זענען אומרעאליסטישע אדער גאר אנדערע פרוכטן, איז דא א שאנס אז א LLM וועט גלויבן א פאלשן ענטפער.

ווי אזוי אגענטן ארבעטן, און דער ערשטער בויען מיט browser use (19:14:03)

האלוצינאציע און די פיר יכולות פון אן אגענט (19:14:03)

דער קורס ענדיגט אפ דעם געדאנק פון פריער: א מאדעל קען אמאל ארויסגעבן אומרעאליסטישע אינפארמאציע וואס קלינגט גלייבווערדיג. דאס רופט מען "האלוצינאציע", און די נייע LLMs האבן שוין א סקאלע צו מעסטן ווי שטארק זיי האלוצינירן.

דערנאך גייט מען איבער צום הויפט-פונקט: ווי אזוי ארבעטן AI-אגענטן? עס ווערט ערקלערט אז אן אגענט ארבעט ווי א פיינער פראבלעם-לייזער מיט פיר עיקר-יכולות:

- פלאנירן
- ארבעטן מיט געצייג (tools)
- זכרון און צוטריט צו וויסן
- דורכפירן די אקציעס

יעדע פון זיי ווערט אויסגעשמועסט באזונדער.

ערשטנס, פלאנירן. אן אגענט הייבט אן מיט א ציל, צ.ב.ש. אויסצופארשן מארק-טרענדס אדער צו שרייבן א בליצבריוו. ער טיילט דעם ציל אויף אין קלארע טריט וואס מען קען דורכפירן, לויט א "chain of thought" שטייגער, כדי די ארבעט זאל גיין אויפן בעסטן אופן. אזוי דארף א מענטש נישט אים יעדעס מאל פון פריער אנצינדן מיט פעסטגעשטעלטע "טריגערס".

צווייטנס, ארבעטן מיט געצייג. אנדערש ווי א פשוט'ער AI-מאדעל, קען אן אגענט ארבעטן מיט געצייג: ער קען ארומבלעטערן אויפן אינטערנעט, פרעגן דאטא-באזעס און ניצן APIs. דאס פארברייטערט זיינע מעגליכקייטן ווייט העכער דאס פעסטע וויסן וואס איז אריין בשעת'ן טריינירן.

דריטנס, זכרון און צוטריט צו וויסן. אן אגענט קען ארויסנעמען און אפהיטן אינפארמאציע דורך טעכניקן ווי RAG (retrieval augmented generation). דורכדעם קען ער אריינברענגען דאטא וואס איז ספעציפיש פאר א פירמע, מארק-פארשונג אדער פראגעס פון קונה-סערוויס, און געבן ענטפערס וואס זענען אקטועל.

פערטנס, און דאס איז וויכטיג: דורכפירן אקציעס. אן אגענט פראצעסירט נישט בלויז אינפארמאציע, ער קען אויך טון: שרייבן באריכטן, פירן אפליקאציעס, אדער אפילו רעדן מיט אנדערע AI-אגענטן. דאס מאכט אים א שטארקער אויטאמאציע-געצייג וואס קען אליין פירן קאמפליצירטע ארבעטס-פראצעסן.

די צוקונפט און די סכנות (19:15:36)

דער קורס זאגט אז AI-אגענטן ברענגען א נייע תקופה פון אויטאמאציע און עפעקטיווקייט, אבער זייער זעלבשטענדיגע נאטור ברענגט מיט זיך סכנות. אן די ריכטיגע שוץ-מיטלען קען אן אגענט פאלש פארשטיין דעם ציל. דער ביישפיל וואס ווערט געבראכט: אויב מען הייסט אן אומקאנטראלירטע AI ברענגען שלום אויף דער וועלט, קען זי אנקומען צו עקסטרעמע לייזונגען. דערפאר וועט זיין קריטיש צו האבן מענטשליכע אויפזיכט, עטישע פראגראמירונג און א פאראנטוואָרטליכן אריינשטעלן, כדי מען זאל קענען ניצן אגענטן זיכער און עפעקטיוו.

צום סוף פון דעם טעארעטישן טייל ווערט צוזאמגעפאסט: AI-אגענטן טוישן ווי אזוי מיר ארבעטן און האבן צו טון מיט טעכנאלאגיע. מיט זייער פעאיגקייט צו פלאנירן, ניצן געצייג, אפהיטן וויסן און דורכפירן ארבעט, זענען זיי ווייט מער פארגעשריטן ווי די געווענליכע LLMs.

די דעמאנסטראציע: אן אגענט וואס בלעטערט אויפן אינטערנעט (19:16:38)

אצינד הייבט זיך אן דער פראקטישער טייל. מען בויט אן אגענט וואס קען אייך העלפן ארומבלעטערן אויפן אינטערנעט. מען ניצט דערפאר browser use, ווייל לויט דעם קורס האט עס די העכסטע געווינקייט קעגן זיינע קאנקורענטן, ארום 89%. עס זענען פאראן דריי פלענער: open source, pro און enterprise. דא נעמט מען די open source ווערסיע, וואס ליגט גרייט אויף GitHub, ווי מען זעט דאס גאנצע repository.

אויף דעם אפיציעלן בלאט פון browser use איז דא א "quick start", און אלע טריט שטייען אין דער ריכטיגער סדר. מען דארף בלויז נאכגיין די טריט כדי צוגרייטן די סביבה פאר'ן בויען דעם אגענט.

אינסטאלירן Python (19:17:39)

מען עפנט דעם command prompt און מען קוקט צו Python איז דא, ווייל דאס איז איינס פון די באדינגונגען. עס ווייזט זיך ארויס אז די ווערסיע איז 3.9, אבער מען דארף כאטש 3.10. דערפאר לאדט מען אראפ א נייע ווערסיע: מען גייט צו דעם Windows-טייל (מען ארבעט אויף Windows) און נעמט Python 3.13.

עס ווערט אויך דערמאנט אז playwright איז איינס פון די באדינגונגען, און אזוי אויך uv. uv איז א Python-פעקל וואס העלפט שאפן א "virtual environment", א באזונדערע סביבה פאר'ן לויפן אייערע אגענטן. ווער עס גייט נאך די טריט קען דאס אליין אויפשטעלן.

ווען Python איז אראפגעלאדן, לויפט מען דעם אינסטאלאציע-טעקע. א וויכטיגע עצה: מען זאל אנקליקן "Add python.exe to PATH". אזוי דארף מען נישט אליין שאפן דעם environment variable, עס ווערט געמאכט אויטאמאטיש. די איבעריגע טריט גייען לויט די default אויסוואלן. נאך דער אינסטאלאציע עפנט מען ווידער א command prompt און מען פרובירט דעם זעלבן באפעל פאר דער ווערסיע. יעצט ווייזט עס 3.13.

אויפשטעלן uv און די virtual environment (19:19:45)

די איבעריגע ארבעט גייט פאר זיך אינעם command prompt, און מען האלט אפן דעם browser use בלאט כדי נאכצוגיין די טריט. מען פרובירט דעם באפעל וואס שטעלט אויף א סביבה מיט uv, אבער מען באקומט א טעות אז uv איז נישט באקאנט (uv is not recognized). מען קוקט אויף Stack Overflow, און מען פארשטייט אז מען דארף קודם אינסטאלירן uv.

אויף דעם אפיציעלן uv בלאט איז דא א דאקומענטאציע מיט טריט. מען קאפירט דעם קאוד פון דארט און לויפט אים אינעם command prompt, און דאס ערשטע פעקל ווערט אראפגעלאדן. דערנאך לויפט מען דעם צווייטן באפעל אין דער רייע (צו בעטן א ספעציפישע ווערסיע דורך א URL), און דערנאך pip install uv. אויף א ריי פלעצער האט מען צופעליג קאפירט דעם פאלשן קאוד, און מען האט עס פארריכט. עס האט אויך אמאל ווידער ארויסגעשפרינגען א טעות, ביז מען האט גענומען דעם Windows-באפעל, און דאן האט עס געארבעט.

יעצט גייט מען צוריק צו דעם quick start פון browser use, קאפירט דעם באפעל וואס שאפט די virtual environment מיט Python 3.13, און לויפט אים. עס נעמט א ביסל צייט, אבער עס ארבעט.

דערנאך דארף מען "אקטיווירן" די virtual environment. אויף Windows טוט מען דאס מיט דעם activate\scripts באפעל וואס שטייט אויפן בלאט, למשל:

activate\Scripts\venv.

א ווארנונג: מען זאל נישט פארגעסן דעם פונקט אין אנהייב. אויב עס לאזט זיך נישט קאפירן, קען מען עס פשוט אריינשרייבן מיט דער האנט. וועז מען איז אינעם virtual environment, זעט מען דעם נאמען אין קלאמערן פארנט פון דער שורה. דאס איז דער סימן אז מען געפינט זיך דארט.

browser use און playwright (19:23:22)

צוריק אויף דעם browser use בלאט, קאפירט מען דעם באפעל וואס אינסטאלירט browser use מיט אלע אנדערע פעקלעך וואס עס דארף. די פעקלעך ווערן אראפגעלאדן און אויסגעפאקט. דער נעקסטער באפעל אינסטאלירט playwright. דאס איז די לעצטע אנגעוויזענע זאך וואס מען דארף אינסטאלירן, און דאס אויך נעמט א ווייל.

ביישפילן פון אגענטן און דאס קלאנירן פונעם repository (19:24:23)

דער לעצטער טריט איז דאס בויען דעם אגענט אליין. עס ווערט ערקלערט אז מען האט צוויי וועגן: מען קען נעמען ביישפילן פון דעם browser use repository, אדער מען קען קאפירן דעם קאוד פון דער אפיציעלער דאקומענטאציע און אים צופאסן ווי מען וויל.

אינעם repository, אונטער examples און דערנאך use cases , געפינט מען א סך פארטיגע אגענטן וואס browser use האט געבויט, למשל:

- אידענטיפיצירן א captcha
- איינקויפן (shopping)
- ארויסזוכן אינפארמאציע פון Wikipedia בלעטער
- סקראלן, און זוכן א געוויס ווארט דורך דעם זוך-קעסטל

מען קען זיי פרובירן און זיך פון זיי לערנען. כדי צו האבן זיי ביי זיך, קלאנירט מען דעם repository: מען קליקט אויף "Code", קאפירט די אדרעס, און לויפט אינעם virtual environment:

git clone <די אדרעס>

אינסטאלירן Git און א PATH פאר אים (19:25:57)

דא באקומט מען א טעות אז git איז נישט באקאנט. מען דארף אינסטאלירן Git. מען עפנט א נייעם command prompt, און מען געפינט א דעטאלירטע אנווייזונג ווי אזוי צו אינסטאלירן Git אויף Windows. מען גייט צו דעם אפיציעלן Git בלאט, לאדט אראפ די 64-bit ווערסיע, און אינסטאלירט לויט די default טריט.

אבער אין א נייעם command prompt איז git נאך אלץ נישט באקאנט. עס ווערט ערקלערט אז אנדערש ווי Python, שטעלט Git נישט אויטאמאטיש אויף דעם PATH , און מען דארף עס אליין צולייגן אינעם environment variables פענצטער. מען דארף נישט נעמען די variables פון דעם באניצער-חשבון, נאר די system variables. דארט קליקט מען "New" און שרייבט אריין דעם וועג צו Git : מען גייט אין דעם Program Files טעקע, דערנאך אין Git, און דערנאך אין cmd. דאס איז דער פלאץ. מען גיט דעם variable דעם נאמען git און לייגט אריין דעם וועג.

דאס האט נישט באלד געארבעט, ווייל לויט די אנווייזונג דארף מען צומאכן און ווידער עפענען דעם command prompt (אדער PowerShell). מען עפנט א נייעם command prompt, אבער מען רירט נישט דעם וואס איז אינעם virtual environment. מען פרובירט דעם באפעל פאר Git'ס ווערסיע, און Git ארבעט שוין.

קלאנירן אינעם virtual environment (19:28:33)

עס ווייזט זיך אז מען דארף ווידער אנהייבן אויך דעם virtual environment. פריער פרובירט מען צו קלאנירן git clone, אין א נייעם command prompt: מען קאפירט די אדרעס, לויפט repository, און דער ווערט קלאנירט. וועו מען זעט אז עס ארבעט, מאכט מען צו אלע command prompts און פרובירט ווידער אינעם virtual environment. מען זעט אז דער טעקע איז שוין דא, הייסט עס אז דאס קלאנירן איז געלונגען.

דער נעקסטער טריט: מען גייט צוריק צום desktop און שאפט א נייעם, ליידיגן טעקע.

אינסטאלירן Web UI און לויפן א בראוזער-אגענט (19:29:33)

אראפנעמען דעם Web UI פאר browser use (19:29:33)

דער פאלדער ווערט איבערגערופן "AI agent". ביז אהער האט מען אויפגעשטעלט אלע סביבות וואס browser use פארלאנגט. כדי צו שאפן דעם AI אגענט דארף מען אבער אויך אן אינטערפעיס, און browser use גיט שוין צו א פארטיגן וועב-באניצער-אינטערפעיס, Web UI. דאס איז דער צווייטער לינק, און מען וועט אים אראפנעמען און אינסטאלירן אינעם זעלבן פאלדער.

מען עפנט ווידער דעם command prompt און מען גייט צוריק צום דעסקטאפ מיטן באפעל cd desktop. דערנאך גיט מען אן דעם פלאץ פונעם פאלדער "AI agent", און מען געפינט זיך שוין אינעווייניג. יעצט דארף מען אויספירן עטליכע באפעלן וואס אינסטאלירן דעם Web UI אין דעם פאלדער.

מען גייט צום היים-בלאט פון browser use Web UI אויף GitHub. דארט געפינט מען א שריט-ביי-שריט אנווייזונג ווי אזוי עס צו אינסטאלירן, און וועו מען פירט עס אויס איז מען גרייט.

- דער ערשטער שריט איז קלאונען. git איז שוין פריער אינסטאלירט געווארן, דעריבער שרייבט מען git clone, א שפאציע און דעם לינק, כדי צו קלאונען דעם גאנצן repository אריין אינעם פאלדער "AI agent".
- צו באשטעטיגן אז עס האט געארבעט, קען מען עפענען דעם פאלדער web-ui אויפן דעסקטאפ און זען אז אלע dependencies זענען דארט.

ווירטועלע סביבה און dependencies (19:31:08)

אין נעקסטן שריט טוישט מען דעם directory צום פאלדער web-ui. אלע ווייטערדיגע שריט ווערן אויסגעפירט פון דארט.

די אנווייזונג זאגט אז מען דארף ווידער ארייננייז אין א ווירטועלע סביבה, און זי רעקאמענדירט צו ניצן UV. UV איז שוין פריער אינסטאלירט געווארן, אבער צוליב זיכערקייט פירט מען אויס דעם באפעל נאכאמאל. עס זענען פאראן צוויי אפציעס, איינע פאר Mac/Linux און איינע פאר Windows, און מען קלייבט Windows. מען קאפירט דעם קאוד, מען לייגט אים אריין אינעם command prompt, און נאך א פאר מאמענטן איז אלעס אינסטאלירט. דערנאך לייגט מען אריין דעם צווייטן קאוד-שטיקל, און מען זעט ווי עס פאקט זיך אויס.

דער דריטער שטיקל שאפט די ווירטועלע סביבה:

- מען קאפירט פונעם GitHub בלאט דעם באפעל uv venv מיט Python 3, מען מעקט אויס דעם פריערדיגן קאוד אינעם פענצטער און מען דריקט Enter.
- דערנאך אקטיווירט מען די venv. אויף Windows קלייבט מען דעם וועג דורך Scripts. פאר Linux און Mac OS איז פאראן אן אנדערער באפעל. מען זעט שוין אז מען געפינט זיך אין דער ווירטועלער סביבה, און אלס נאמען ווייזט זיך web-ui.
- דערנאך אינסטאלירט מען די dependencies מיט uv pip install -r requirements.txt. מען קאפירט און מען לייגט אריין דעם קאוד, און די dependencies הייבן אן אראפצוקומען. עס נעמט א ביסל צייט ביז זיי ווערן אראפגענומען און אויסגעפאקט.
- נאכדעם אינסטאלירט מען ווידער Playwright. מען קאפירט דעם באפעל פונעם GitHub בלאט.

Docker אדער לאקאלע אויפשטעלונג (19:33:49)

דער נעקסטער שריט אין דער אנווייזונג איז Docker, אבער מען קלייבט א לאקאלע אויפשטעלונג. דער חילוק ווערט ערקלערט אזוי: אויב מען גיט איבער דעם AI אגענט צו אן אנדערן באניצער אדער מען מאכט דעם אגענט עפנטלעך, קען מען ניצן Docker. ווען מען שאפט אבער אן אגענט נאר פאר זיך און פאר די אייגענע צוועקן, ניצט מען א לאקאלע אויפשטעלונג, און דער אגענט וועט זיין אקטיוו נאר אויף דעם אייגענעם קאמפיוטער אין א לאקאלער סביבה.

נאכדעם וואס Playwright איז אינסטאלירט, פירט מען אויס דעם באפעל וואס לאזט לויפן דעם באניצער-אינטערפעיס. דער command prompt גיט צוריק אן IP אדרעס אין דער פארעם פון localhost. מען ווארט א ווייל ביז עס ווייזט זיך "running local on URL". די אדרעס איז http://127.0.0.1 מיט port נומער 7788. מען קאפירט די URL און מען לייגט זי אריין אין סיי וועלכן לאקאלן בראוזער אין א נייעם פענצטער. מען זעט אז Web UI לויפט מיט הצלחה מיט אן אינטערפעיס פארן אגענט.

איינשטעלונגען, LLM און API שליסל (19:35:21)

דער אגענט איז נאך נישט גאנץ פארטיג. מען זעט איינשטעלונגען ווי מאקסימום שריט פאר א לויף (max run steps) און מאקסימום אקציעס פער שריט. ווען מען גיט א באפעל, לויפט ער א געוויסע צאל שריט, און מען קען די צאל פארגרעסערן אדער פארקלענערן לויט ווי מען ווייל. אין דער לעקציע שטעלט מען די מאקסימום אקציעס אויף 15.

דער וויכטיגסטער טייל איז די LLM Settings. לויט דיפאלט שטייט דארט OpenAI , אבער מען קלייבט Google, און דארט דעם מאדעל Gemini 2.0 Flash. דערצו דארף מען אן API שליסל:

- מען עפנט א נייעם טאב און מען זוכט Google AI Studio, דעם ערשטן לינק. מען דארף זיך אריינלאגירן. עס ווערט געוואגט או מען דארף זיך נישט זארגן וועגן פרייז, ווייל עס איז פריי און קאסט נישט קיין געלט.
- נאכן אריינלאגירן קלייבט מען Get API key. מען מאכט צו דעם פראמפט וואס ווייזט זיך, מען נעמט אן אלע תנאים, מען דריקט נאכאמאל Get API key, מען סקראלט אראפ און מען דריקט Create API key, און דער שליסל ווערט געשאפן.
- א ווארענונג: דער API שליסל איז נאר פאר אייער פריוואטן באנוץ, און מען טאר אים מיט קיינעם נישט טיילן. אפילו דאס פענצטער אליין זאגט אזוי.

• מען קאפירט דעם שליסל און מען לייגט אים אריין אינעם Web UI.

ערשטער לויף: זוכן Simplilearn (19:37:27)

יעצט קען מען לאזן לויפן דעם אגענט און אריינשרייבן די שריט וואס ער זאל נאכפאלגן. דער פראמפט איז: גיי צו google.com און זוך דעם simplilearn.com וועבזייטל.

אינעם command prompt זעט מען די שריט-ביי-שריט אויספירונג. דער אגענט עפנט א נייעם Google בראוזער-פענצטער אין "signed out" מאדע, נעמט אליין דעם זוך-קעסטל און שרייבט אריין, אן דעם וואס דער באניצער זאל עפעס טייפן. דערנאך דריקט ער אליין דעם Enter אדער זוך-קנעפל, דערקענט די זוך-רעזולטאטן, דערקענט דעם ריכטיגן בלאט און עפנט דעם היים-בלאט פון Simplilearn. מען קען אים אויך הייסן געפינען א קורס וואס מען זוכט.

צווייטער פרואוו: Gmail מיט לאגין-פרטים (19:38:29)

מען שטעלט אפ די אפליקאציע און מען גייט א שריט ווייטער. מען גיט דעם אגענט אן אימעיל אדרעס און לאגין-פרטים, און מען בעט אים ער זאל זיך אריינלאגירן און ארויסשיקן אן אימעיל. דער פריערדיגער לויף איז דאך געווען אין "signed out " מאדע, און יעצט זאל דער אגענט עפענען א נייעם פענצטער, אריינלאגירן אין Gmail און שיקן אן אימעיל.

מען מעקט אויס דעם פריערדיגן פראמפט און מען שרייבט אזוי:

• שריט 1: גיי צו google.com.

• שריט 2: לאגיר אריין אין Gmail מיט אן אימעיל אדרעס און א פאסווארט.

• שריט 3: שרייב אן אימעיל צו א צופעליגן אדרעס מיטן subject "Enrolled to AI and ML program".

דער אגענט עפנט א נייעם פענצטער, דערקענט און קלייבט דעם sign in קנעפל, עפנט דעם לאגין-בלאט און פרובירט אריינצוטייפן. עס ווערט באמערקט אז צוליב געוויסע זיכערהייט-באגרענעצונגען זאל ער עס נישט קענען דורכפירן, הגם מען זעט דעם קורסאר באלויכטן אינעם פענצטער. צום סוף שטעלט ער זיך אפ פון זיך אליין.

די מסקנא איז אז דער אגענט האט באגרענעצונגען, אבער ער איז סך-הכל גוט. פאר בוקן זאכן אויף אן אנליין וועבזייטל, זוכן די בעסטע פראדוקטן פאר נידעריגן פרייז, אדער אנדערע באפעלן וואס זענען נישט קיין זיכערהייט-פראגע, איז דער אגענט זייער גוט. אבער וועז מען גייט א שריט ווייטער צו עפעס וואס איז עטיש נישט לאגיש צו טון, ציט זיך דער אגענט צוריק.

צוגרייטונגען פאר אן OpenAI פראיעקט: Python און ביבליאטעקן (19:41:01)

א נייער טייל הייבט אן. פארן אנהייבן דארף מען זיין זיכער אז Python איז אינסטאלירט, מינימום וועריסיע 3.7. מען קען אראפנעמען די לעצטע וועריסיע פון python.org/downloads. צו קאנטראלירן: Windows+R, cmd, און מען שרייבט python --version. די וועריסיע אין דער לעקציע איז 3.13.7, אבער יעדע וועריסיע פון 3.7 און העכער ארבעט.

דערנאך אינסטאלירט מען עטליכע Python ביבליאטעקן:

• pip install openai, וואס אינסטאלירט אלע נויטיגע פעקלעך.

• pip install requests, אן אלגעמיינע ביבליאטעק פאר HTTP בקשות.

- pip install flask און pip install fastapi, ווייל מען ניצט Flask און FastAPI.
- python-dotenv, כדי צו האלטן דעם שליסל אין א .env טעקע און אים לייענען פון פראגראם. דער ערשטער פרואוו האט געגעבן א פעלער צוליב א טעות אינעם נאמען, און דער ריכטיגער באפעל איז pip install python-dotenv. ווי‌באלד עס איז שוין פריער אינסטאלירט געווען, ווייזט זיך "requirement already satisfied".

Git און דער OpenAI API שליסל (19:42:33)

Git איז אויך נויטיג, פאר ווערסיע-קאנטראל און צו קלאונען repositories. מען נעמט עס אראפ פון git-scm.com. דארט קלייבט מען די Windows אינסטאלאציע, מען פאסט אויף אז זי זאל פאסן צו דער אייגענער Windows ווערסיע, און מען נעמט זי אראפ. צו קאנטראלירן שרייבט מען git --version , און מען זעט 2.52.0.windows.1, א סימן אז git איז גרייט.

צו ניצן OpenAI מאדעלן דארף מען אן API שליסל, און מען קען ניצן OpenAI אדער אפילו Google:

- מען גייט צו platform.openai.com, מען שאפט א חשבון מיט אן אימעיל און א גוטן פאסווארט.
- אויף דער לינקער זייט קלייבט מען API keys, מען דריקט דעם פלוס-קנעפל, און מען גיט א נאמען (אין דער לעקציע "test") און א פראיעקט ("default"). דערנאך דריקט מען Create secret key און מען קאפירט דעם שליסל.

שאפן דעם פראיעקט-פאלדער, .env און app.py (19:44:05)

יעדער פראיעקט ווערט געהאלטן אין א פאלדער. מען שאפט א נייעם פאלדער, למשל "default" אדער "test", און מען עפנט אים אין דער קאוד-רעדאקטאר. דער ערשטער זאך וואס מען שאפט איז א .env טעקע, וואס ווערט גענוצט ספעציעל צו האלטן API שליסלען. דארט שרייבט מען דעם נאמען פונעם שליסל אין גרויסע אותיות, אן "=" צייכן, און דעם שליסל וואס מען האט קאפירט פון OpenAI, למשל:

OPENAI_API_KEY=...

מען היט אפ די טעקע. דערנאך שאפט מען א Python טעקע מיטן נאמען app.py.

ערשטע אגענטן מיט OpenAI, Gemini און דרויסנדיגע API (19:45:11)

א פשוטער פראגראם מיט דעם OpenAI שליסל (19:45:11)

דער קורס ווייזט א פשוטן פראגראם וואס ניצט דעם OpenAI API שליסל. דער ערשטער שריט איז אריינצוברענגען (import) די פונקציע load_dotenv: from dotenv import load_dotenv. עס ווערט ערקלערט אז די שורה איז געמיינזאם פאר אלע סארטן פראגראמען, ווייל מען לאדנט די שליסלען פון דער .env טעקע.

דערנאך ברענגט מען אריין openai, ווייל דאס איז וואס מען ניצט דא. פאר Google ניצט מען א צווייטן סטיל פון פראגראמירן, און דער קורס קומט צוריק צו דעם שפעטער. import openai ברענגט אריין דעם OpenAI Python SDK, כדי מען זאל קענען מאכן API רופן.

דערנאך קומט import os. דאס ברענגט אריין דעם os מאדול, כדי צו דערגרייכן די "environment variables" פון דער סיסטעם.

• load_dotenv() לייענט די .env טעקע און לאדנט אלע וועריעבלס פון איר.

• openai_api_key = os.getenv(...): אין די קלאמערן שרייבט מען דעם נאמען וואס מען האט געגעבן דעם שליסל אין דער .env טעקע. עס איז נישט דער נאמען פון דער טעקע, נאר דער נאמען אין דער שורה "API key = אזוי און אזוי". די שורה ברענגט דעם וועדט פונעם שליסל פון דער סביבה.

• openai_api_key = openai.api_key: די שורה שטעלט איין דעם שליסל, כדי דער SDK זאל קענען אויטאמאטיש שיקן בקשות.

א ביישפיל פון אן API רוף צו OpenAI (19:46:47)

דערנאך ווייזט מען א ביישפיל פון אן API רוף:

response = openai.Completion.create(engine="text-davinci-003", prompt="Say this is a test", max_tokens=5)

עס וועדט ערקלערט וואס יעדער טייל טוט:

• די שורה רופט דעם OpenAI API צו שאפן א טעקסט-השלמה (completion).

• דער engine באשטימט וועלכער מאדעל וועדט גענוצט. דא איז דאס text-davinci-003, אבער מען קען נעמען וועלכן מען וויל.

• דער פראמפט איז "Say this is a test".

• max_tokens=5 באגרעניצט דעם ענטפער צו מאקסימום פינף טאקענס אדער וועדטער.

צום סוף דרוקט מען דעם ענטפער מיט print(response.choices...). דאס דרוקט דעם טעקסט וואס דער AI האט צוריקגעגעבן, אן איבעריגע ספעיסעס.

נאך מכשירים און די סטרוקטור פונעם פראיעקט (19:47:52)

נאכדעם וואס מען האט דעם API שליסל, קען מען, וועז מען וויל, אראפלאדן כלים פאר קלאוד אדער פאר לאקאלער אנטוויקלונג. למשל Docker: אויב מען שטעלט ארויס דעם AI אגענט אין א "container" סביבה, דארף מען האבן Docker אינסטאלירט. חוץ Docker קען מען אויך ניצן Heroku, AWS אדער Google Cloud.

דערנאך ערקלערט דער קורס ווי מען שטעלט אויף דעם פראיעקט-פאלדער, אויב מען האט אים נאך נישט געשאפן. די טעקעס וואס דארפן זיין:

• .env – האלט דעם OpenAI API שליסל.

• א Python טעקע וואס לויפט דעם הויפט-סקריפט, דא app.py.

• (וועז מען וויל) requirements.txt – א טעקסט טעקע מיט אלע באדערפענישן. מען לויפט זי, און מען באקומט אלע Python ביבליאטעקן און אנדערע ביבליאטעקן וואס מען דארף.

• (וועז מען וויל) README.md – די דאקומענטאציע פונעם פראיעקט. דארט קען מען פארשרייבן, בשעת מען ארבעט, וועלכע ענדערונגען מען האט געמאכט אין פראגראם, אדער סיי וועלכע אינפארמאציע איבערן פראיעקט.

א פרייער שליסל פון Google Gemini (19:48:55)

וויבאלד דער שליסל פון OpenAI קאסט געלט, ווייזט דער קורס אן אנדער וועג: א Google API שליסל, וואס איז פריי. די שריט:

- מען גייט צו ai.google.dev (דער Gemini API) און מען עפנט א חשבון מיט בלויז אן אימעיל און א פאסווארט.
- מען קליקט אויף "Get API key". דארט זעט מען די פריערדיגע שליסלען.
- כדי צו שאפן א נייעם, קליקט מען "Create API key", מען גיט א נאמען צום פראיעקט (למשל chatbot), און מען שאפט דעם שליסל.
- מען קאפירט אים מיט דער אפציע "Copy API key".

עס ווערט געווארנט: פאר זיכערהייט, טיילט נישט אייער API שליסל מיט אנדערע, ווייל זיי קענען אים מיסברויכן.

דערנאך גייט מען צוריק צו דער .env טעקע. אנשטאט דער שורה פון דעם OpenAI שליסל, שרייבט מען: GEMINI_API_KEY= און מען קלעבט אריין דעם שליסל. דערנאך היט מען אפ די טעקע. דער שליסל וועט געגנוצט ווערן איבערן גאנצן קאוד, ווייל מען קען רופן קליינע פונקציעס וואס דערגרייכן אים.

אינסטאלירן Google Generative AI (19:50:28)

נאכן שליסל איז דער וויטערדיגער וויכטיגער שריט צו אינסטאלירן Google Generative AI:

- מען עפנט דעם Command Prompt.
- מען גייט אריין אין פאלדער מיט cd און דעם ארט, און מען דריקט Enter.
- מען לויפט: pip install google-generativeai. אין דער לעקציע גיט מען צו א געוויסע ווערסיע, אבער מען קען נעמען די לעצטע ווערסיע.
- כדי צו פארזיכערן אז עס איז טאקע אינסטאלירט, לויפט מען pip show google-generativeai. עס ווערן געוויזן די ווערסיע, א קורצע באשרייבונג (summary), די היים-בלאט (homepage), דער מחבר (author) א.א.וו.

דער ערשטער Gemini אגענט אין app.py (19:51:34)

דער הויפט-פראגראם גייט אין app.py. דער קאוד, שריט נאך שריט:

- import google.generativeai as genai
- from dotenv import load_dotenv
- import os
- load_dotenv() – לאדנט דעם שליסל.
- api_key = os.getenv("GEMINI_API_KEY") – דערגרייכט דעם ווערט פונעם שליסל פון די environment variables.

- א רעזערוו-פלאן: אויב עס איז נישטא קיין שליסל (if not api_key), דרוקט מען א פעלער-מעלדונג "Error: No API key found" און מען גייט ארויס (exit). אזוי ווערט געדרוקט א מעלדונג און דער פראגראם הערט אויף, ווען עס פעלט א שליסל.
- genai.configure(api_key=api_key) – שטעלט איין דעם Gemini API קליענט.
- model = genai.GenerativeModel("models/gemini-2.5-flash") – באשטימט וועלכן מאדעל צו רופן. דער קורס נעמט Gemini 2.5 Flash און באצייכנט אים ווי דעם לעצטן.

דערנאך דעפינירט מען א פונקציע chat_with_agent(prompt) מיט א try/except:

- try: response = model.generate_content(prompt). דאס שאפט אינהאלט פונעם געגעבענעם פראמפט. דערנאך דרוקט מען "AI Response:" מיט response.text.
- except Exception as e: דרוקט מען דעם פעלער.

צום סוף קומט דער הויפט-בלאק: if name == "main": מען רופט chat_with_agent מיטן פראמפט "Tell me a funny AI joke in one sentence". דער AI גייט דורך דעם זאץ און גיט צוריק דעם ענטפער.

לויפן דעם קאוד און פארריכטן פעלערן (19:54:49)

מען לויפט דעם קאוד מיט python און דעם נאמען פון דער טעקע (app.py). דער ערשטער לויף האט געוויזן "invalid syntax" אינעם הויפט-בלאק. די סיבה איז געווען אז עס האבן געפעלט די טאפלטע אונטערשטריכן ארום name און main. נאכן פארריכטן האט מען ווידער געלאפן דעם פראגראם.

דער Gemini מאדעל האט געענטפערט מיט א וויץ איבער AI: פארוואס האט דער AI זיך צעשיידט מיטן קאלקולאטאר? ווייל ער האט געפילט אז דער קאלקולאטאר "לייגט נאר צו" צו זיינע פראבלעמען.

א עצה פונעם קורס: אויב קומט ארויס א פעלער, קאפירט און קלעבט אריין דעם פעלער אין ChatGPT אדער Gemini. זיי וועלן אנווייזן וועלכע תיקונים מען דארף מאכן און פארוואס דער פעלער איז פארגעקומען. אזוי פארשטייט מען וואס דער פראבלעם איז, און ווי אזוי אים צו לייזן.

דרויסנדיגע API רופן: דאטום און צייט (19:55:50)

ויבאלד דער ערשטער אגענט ארבעט שוין, גייט מען א שטאפל טיפער. א AI אגענט ווערט שטארק ווען ער קען ברענגען דאטא פון דער ריכטיגער וועלט: וועטער, צייט, קאלענדאר, געשעענישן, אימעילס אדער אפילו אקציע-פרייזן. דא לערנט מען ווי אזוי צו מאכן א רוף צו אן דרויסנדיגן API פונעם אגענט.

דער ערשטער שריט איז pip install requests . מען האט עס שוין געטון, אבער מען לויפט עס נאכאמאל. די requests ביבליאטעק לאזט Python פראגראמען רעדן מיט סיי וועלכן דרויסנדיגן API, סיי Google Calendar, סיי וועטער, סיי א פריוואטער "back end".

מען שאפט א נייע Python טעקע אינעם זעלבן פאלדער, api_tools.py, און מען שרייבט א פראגראם וואס גיט צוריק דאטום און צייט:

- import requests
- מען דעפינירט א פונקציע get_current_time(), דערווייל אן פאראמעטערס.
- מען גיט א URL פון א API "endpoint", וואס גיט צוריק די איצטיגע צייט פאר דער צייט-זאנע Asia/Kolkata.

- requests.get(url) = response – שיקט אן HTTP בקשה און באקומט די צייט ווי אן ענטפער.
- response.raise_for_status() – ווארפט א פעלער אויב דער API רוף איז דורכגעפאלן, למשל ווען מען קען נישט באקומען די צייט.
- response.json() = data – פארוואנדלט דעם JSON ענטפער אין א Python דיקשאנערי.
- data["datetime"] – גיט צוריק דעם דאטום און די צייט.

א וויכטיגע ווארענונג: לויפט נישט די טעקע גלייך. די פונקציע שיקט אן HTTP GET בקשה צו א פובליקן צייט-API, באקומט א JSON ענטפער, און ציט ארויס די איצטיגע צייט. אזוי ווען דער אגענט ארבעטן מיט סיי וועלכע מקורות פון ריכטיגע דאטא.

פאראייניגן דעם API מיטן אגענט (19:58:26)

מען שאפט נאך א טעקע, agent_api_call.py, צו פאראייניגן דעם API מיטן אגענט. דער קאוד:

- מען ברענגט אריין google.generativeai און dotenv, ווי אין יעדן Python פראגראם.
- from api_tools import get_current_time – ברענגט אריין די אייגענע פונקציע וואס ברענגט דאטא אין ריכטיגער צייט.
- load_dotenv() – לאדנט אלע ווערייעבלס פון דער .env טעקע אין דער סביבה.
- genai.configure(...) – שטעלט איין דעם Gemini API מיטן שליסל פון דער סביבה.
- model – ווידער Gemini 2.5 Flash.
- מען דעפיניירט א פונקציע agent_with_api(prompt), וואס באארבעט דעם אינפוט פונעם באניצער. למשל, ווען דער באניצער פרעגט עפעס וואס פאדערט אן API, רופט זי דעם API.
- if "time" in prompt: – דאס קוקט אויב דאס ווארט "time" קומט פאר אין דער פראגע פונעם באניצער. עס ווערט באמערקט אז דער טשעק הייבט אכט אויף גרויסע און קליינע אותיות.
- real_time = get_current_time() – רופט די דרויסנדיגע API פונקציע צו באקומען דאטא אין ריכטיגער צייט.
- דערנאך ווערט דער פראמפט אויסגעברייטערט: מען לייגט צו צום prompt (prompt + ...) די אינפארמאציע וואס מען האט באקומען.

אגענט מיט לאנג-טערמיניגן זכרון און פראמפט-קייטן (20:00:33)

ענדיגן דעם אגענט וואס רופט אן א דרויסנדיגע API (20:00:33)

דער קורס ענדיגט דעם קאוד וואס ווייזט ווי אן אגענט באנוצט זיך מיט אן אינדרויסנדיגע API כדי צו ברענגען אמת'ע לייוו-דאטא. אין דעם פראמפט ווערט צוגעלייגט א שורה וואס זאגט, אז "current time" איז גלייך צו דער לייוו-דאטא. אזוי ווערט די צייט וואס מען האט ארויסגעצויגן אריינגעלייגט אינעם פראמפט, און דער AI מאדעל באקומט עקסטערע קאנטעקסט.

דערנאך קומט די שורה response = model.generate_content(prompt). זי רופט אן Gemini, אז ער זאל שאפן אן ענטפער לויטן פארבעסערטן פראמפט. נאכדעם קומט return response.text, וואס גיט צוריק דעם טעקסט-ענטפער צום באניצער.

צום סוף איז דא דער main טייל פונעם פראגראם. ער מאכט זיכער, אז דער קאוד לויפט נאר וועז מען גיט א פראמפט. אין קאוד איז שוין אריינגעשטעלט א פראמפט פון פריער: "What is the current time right now". דעם פראמפט קען מען טוישן צו וואס מען ווייל.

דער קורס פאסט צוזאם וואס מען האט דא געטון:

- אויב דער באניצער'ס מעסעדזש אנטהאלט דאס ווארט "time", רופט מען אן א דרויסנדיגע API וואס הייסט get_current_time.
- מען לייגט אריין די אמת'ע וועלט-דאטא אינעם פראמפט פונעם AI מאדעל.
- דער אגענט ענטפערט מיט ביידע: מיט זיין אייגענעם שכל און מיט די דרויסנדיגע אינפארמאציע.

וועז מען לויפט דעם טייל פונעם פראגראם (דעם agent API call), זעט מען די איצטיגע דאטום און צייט, לויטן פראמפט.

פארוואס דארף אן אגענט זכרון (20:01:34)

ביז אהער איז דער אגענט געווען אינגאנצן "stateless", דאס הייסט אן א מצב. ער ענטפערט יעדן פראמפט באזונדער, און נאכן ענטפערן פארגעסט ער אלעס. דער קורס ערקלערט, אז אמת'ע קלוגע אגענטן דארפן האבן זכרון. א זכרון לאזט דעם אגענט:

- געדענקען וואס דער באניצער האט ליב און וויל;
- זיך דערמאנען אין פריערדיגע אויפגאבעס;
- נאכפאלגן ארבעטן וואס האבן אסאך טריט;
- פארשטיין לאנג-טערמיניגן קאנטעקסט.

מען בויט איצט א לאנג-טערמיניגן זכרון פארן אגענט. עס ווערט אנגעווויזן אויפן אונטערשייד:

- קורץ-טערמיניגער זכרון עקזיסטירט נאר בשעת איין לויף, און ער ווערט אויסגעמעקט וועז דער פראגראם הערט אויף.
- לאנג-טערמיניגער זכרון בלייבט אויף שטענדיג. דער אגענט קען זיך אויסלערנען וועגן אייך און עס געדענקען איבער עטליכע סעסיעס, אפילו נאכדעם וואס מען פארמאכט די אפליקאציע.

די זכרון-סיסטעם באנוצט זיך מיט א גאר איינפאכע און פארלעסליכע סטרוקטור:

- א JSON פייל, agent_memory.json, וואס היט אפ אלע פריערדיגע שמועסן.
- א "memory manager", וואס לייענט פונעם פייל און שרייבט אריין אין אים.
- דער אגענט נעמט ארויס די לעצטע דריי ביז פינף אינטעראקציעס פאר קאנטעקסט.
- דער זכרון וואקסט מיט דער צייט, פונקט ווי א מענטשליכער מוח.

מען דארף שאפן דריי Python פיילן:

- memory_manager.py – היט אפ און נעמט ארויס דעם זכרון.
- agent_with_memory.py – די לאגיק פונעם AI אגענט וואס באנוצט זיך מיטן זכרון.
- test_memory_agent.py – לויפט דעם גאנצן שמועס.

דער ערשטער פייל: memory_manager.py, דער "מוח-שפייכלער" (20:03:06)

דער קורס באצייכנט דעם פייל אלס א "brain storage", א שפייכלער פונעם מוח. זיין צוועק איז אפצוהיטן און ארויסצונעמען דעם לאנג-טערמיניגן זכרון. ער אנטהאלט:

- קאוד וואס שאפט דעם זכרון-פייל agent_memory.json;
- א פונקציע וואס היט אפ די אינטעראקציעס;
- א פונקציע וואס לאדנט ארויף דעם פריערדיגן זכרון.

דער פייל איז דא, ווייל דער זכרון דארף זיין אומאפהענגיג פון דער לאגיק פונעם אגענט. ער איז ווי די דאטא-באזע פונעם אגענט.

דער קאוד, טריט ביי טריט:

- import json – ברענגט אריין דעם JSON מאדול, כדי צו קענען לייענען און שרייבן.
- import os – ברענגט אריין דעם OS מאדול, כדי צו ארבעטן מיט פיילן.
- MEMORY_FILE – דער נאמען פונעם JSON פייל וואו דער זכרון ווערט אפגעהיט.
- דערנאך קומט א קאוד וואס שאפט דעם פייל אויב ער עקזיסטירט נאך נישט:

:if not os.path.exists(MEMORY_FILE)

:as f ("w" ,with open(MEMORY_FILE

(f ,{[] :"history"}json.dump(

דער קאוד קוקט צי דער פייל עקזיסטירט שוין. אויב נישט, שאפט ער אים אין שרייב-מאדע ("w") און שרייבט אריין א ליידיגע JSON סטרוקטור. דער json.dump שטעלט אויף א ליסטע וואו די זכרון-רעקארדס וועלן ווערן אפגעהיט. דאס הייסט: אויב עס איז נישטא קיין JSON פייל אין דעם פאלדער וואו דער פראגראם ליגט, שאפט מען א ליידיגן, און שפעטער קען מען אריינלייגן אין אים אלע אינטעראקציעס.

די פונקציע save_to_memory באקומט צוויי זאכן: דעם טעקסט פונעם באניצער און דעם ענטפער פונעם אגענט. זי היט אפ ביידע מעסעדזשעס אינעם לאנג-טערמיניגן זכרון, אינעם JSON פייל וואס מען האט געשאפן (אדער וואס ווערט געשאפן אויטאמאטיש). די טריט:

- with open(MEMORY_FILE, "r") as f – עפנט דעם פייל אין לייען-מאדע.
- data = json.load(f) – לאדנט דעם JSON אינהאלט אריין אין א Python dictionary.
- data["history"].append(...) – לייגט צו א נייעם זכרון-איינטראג, א dictionary, צו דער history ליסטע. אין אים איז דא "user" מיט user_text, וואס היט אפ וואס דער באניצער האט געשריבן, און "agent", וואס היט אפ דעם ענטפער פונעם אגענט.
- מען עפנט נאכאמאל דעם פייל אין שרייב-מאדע ("w") מיט json.dump(data, f, indent=4). דאס שרייבט צוריק דעם דערהיינטיגטן זכרון אין פייל, און דערנאך דרוקט מען עס אויס.

די צווייטע פונקציע, get_recent_memory, נעמט ארויס דעם לעצטן זכרון. זי עפנט דעם פייל, לייענט די עקזיסטירנדע דאטא, לאדנט דעם JSON אריין אין א Python dictionary, און גיט צוריק די לעצטע n שמועסן וואס זענען פארגעקומען.

דער צווייטער פייל: agent_with_memory.py, דער "AI מוח" (20:06:11)

דער פייל ווערט באצייכנט אלס דער "thinking unit", דער טייל וואס טראכט. ער באשטימט ווי דער אגענט באנוצט זיך מיטן זכרון וועו ער ענטפערט דעם באניצער. ער טוט די פאלגנדע זאכן:

- ער נעמט ארויס די לעצטע n זכרונות פון memory_manager.py;
- ער פארבינדט דעם זכרון מיט דער פראגע פונעם באניצער;
- ער שיקט דעם פארבעסערטן פראמפט צו Gemini;
- ער היט אפ דעם נייעם זכרון, ווידער דורך memory_manager.py.

די אינפארמאציע וואס דער פייל פראצעסירט: דער פריערדיגער זכרון, די איצטיגע פראגע פונעם באניצער, די AI ענטפערס, און דער דערהיינטיגטער זכרון. דער פייל איז דא, ווייל דער אגענט דארף א קלארן ארט וואו מען באהאנדלט דאס טראכטן, דאס בויען קאנטעקסט און דאס שאפן פראמפטס. ער איז ווי דער מוח פונעם AI.

צום אנהייב ברענגט מען אריין די געוועלכע לייברעריס: google.generativeai as genai, dotenv און os. פון memory_manager ברענגט מען אריין save_to_memory און get_recent_memory . דערנאך קומען load_dotenv(), דער API שליסל און genai.configure(api_key=...), גלייך ווי אין די פריערדיגע פראגראמען. דער מאדעל איז דער זעלבער, 2.5 Flash.

די הויפט-פונקציע, ai_agent_with_memory(user_input):

- past_memory = get_recent_memory() – לאדנט די לעצטע דריי אינטעראקציעס, די לעצטע אפגעהיטענע שמועס-איינטראגן.
- memory_text = "" – שאפט א ליידיגן string, וואו מען וועט אנזאמלען דעם צוגעגרייטן זכרון-טעקסט.
- א past_memory loop: for item in, וואס לייגט צו ביי יעדן איינטראג ביידע, דעם באניצער'ס מעסעדזש (item["user"]) און דעם אגענט'ס ענטפער (item["agent"]), צום memory_text.
- final_prompt – אן f-string וואס הייבט זיך אן מיט "You are an AI agent with memory. Here is your past memory:". דא לייגט מען אריין די שמועס-היסטאריע, און דאס איז דער אנהייב פונעם פראמפט. דערנאך קומט "Now answer the new user message:" מיט דער נייער פראגע פונעם באניצער.
- response = model.generate_content(final_prompt) – שיקט דעם צוואמגעשטעלטן פראמפט צו Gemini.
- answer = response.text – נעמט ארויס דעם טעקסט פונעם מאדעל.
- save_to_memory(user_input, answer) – היט אפ דעם לעצטן מעסעדזש און ענטפער, און דערהיינטיגט דעם JSON פייל, כדי דער אגענט זאל עס שפעטער געדענקען.
- צום סוף: return answer.

דער דריטער פייל: test_memory_agent.py, דער שמועס-אינטערפעיס (20:09:21)

דאס איז דער פייל וואס מען לויפט טאקע. ער אנטהאלט א loop וואס פרעגט דעם באניצער וואס ער וויל שרייבן, גיט דאס איבער צו agent_with_memory, וועיזט דעם ענטפער, און האלט אן דעם שמועס. ער

באהאנדלט די מעסעדזשעס פונעם באניצער און די AI ענטפערס, אבער ער אליין היט נישט אפ קיין זכרון. די ארבעט פונעם אפהיטן לאזט ער איבער פאר די אנדערע פיילן, אזוי ווי דער JSON פייל. דער פייל איז דא, ווייל מען דארף א ריינעם און איינפאכן אריינגאנג כדי צו פרובירן דעם אגענט.

דער קאוד:

- from agent_with_memory import ai_agent_with_memory – ברענגט אריין די הויפט-פונקציע, וואס באהאנדלט דעם זכרון, בויט דעם פראמפט, שאפט דעם ענטפער און היט אפ דעם שמועס.
- א print מיט "(exit' to quit' AI Agent with Memory (type" – א באגריסונג, וואס ערקלערט אויך ווי אזוי ארויסצוגיין.
- while True – אן אומענדליכער loop, וואס האלט אן דעם שמועס.
- user_message – נעמט אריין דעם אינפוט פונעם באניצער יעדעס מאל ווען ער שרייבט עפעס.
- אויב דער מעסעדזש איז "exit", ברעכט מען אויף דעם loop. אנדערש: reply = ai_agent_with_memory(user_message). דאס שיקט דעם מעסעדזש צום אגענט, וואס ענטפערט און דערהיינטיגט דעם זכרון.
- צום סוף דרוקט מען אויס דעם ענטפער אויפן עקראן.

דער גאנצער שטראם, ווי דער קורס פאסט עס צוזאם:

- דער באניצער שרייבט אין test_memory_agent.py.
- דער פייל שיקט דעם מעסעדזש צו agent_with_memory.py.
- agent_with_memory.py בעט דעם פריערדיגן זכרון פון memory_manager.py.
- memory_manager.py גיט צוריק דעם זכרון.
- agent_with_memory.py פארבינדט דעם זכרון מיט דער פראגע, שאפט אן ענטפער, און היט אפ דעם נייעם זכרון.
- דער ענטפער גייט צוריק צו test_memory_agent.py, וואו מען ווייזט אים דעם באניצער.

דעמאנסטראציע: דער אגענט געדענקט (20:12:25)

מען גייט אריין אין פאלדער און לויפט python test_memory_agent.py. עס קומט ארויף די באגריסונג. דער שמועס גייט אזוי:

- מען שרייבט "Hi, my name is Susan". דער אגענט ענטפערט, אז ער וועט געדענקען אז דער נאמען איז Susan.
- מען שרייבט "My age is 32". דער אגענט געדענקט אויך דעם עלטער.
- מען שרייבט, אז מען ארבעט אינעם פעלד פון data science. איצט געדענקט ער דריי פרטים.
- מען פרעגט: "What's my name, age and domain that I work in?" דער ענטפער: דער נאמען איז Susan, דער עלטער איז 32 יאר, און דאס פעלד איז data science.

דער אגענט געדענקט טאקע די לעצטע דריי שמועסן. מען קען פארגרעסערן די צאל שמועסן וואס ער געדענקט, לויט ווי מען וויל.

דריי נייע כוחות: פראמפט-קייטן, ראוטינג און פאראלעלע ארבעט (20:13:27)

ביז איצט קען דער אגענט געדענקען אינפארמאציע, אנרופן דרויסנדיגע APIs און שאפן ענטפערס. איצט לייגט מען צו דריי שטארקע פעאיקייטן:

- prompt chaining, א קייט פון פראמפטס;
- routing, אויסקלייבן וואוהין צו שיקן א בקשה;
- parallel task execution, אויספירן אויפגאבעס גלייכצייטיג.

אזעלכע טעכניקן באנוצן זיך אמת'ע AI אסיסטענטן אין דער וועלט, כדי צו צעטיילן אן אויפגאבע אויף שטיקלעך, אויסקלייבן די ריכטיגע געצייג, און ארבעטן מער עפעקטיוו.

ביי prompt chaining פרובירט דער אגענט נישט אפצוטון אלעס מיט איין מאל. ער מאכט עטליכע טריט, און דער רעזולטאט פון איין טריט ווערט דער אינפוט פונעם קומענדיגן. דער ביישפיל: מען שרייבט "Explain Python decorators and give an example". אן א פראמפט-קייט פרובירט דער אגענט צו ענטפערן אלעס מיט איין מאל, און דאס קומט אפט ארויס צעמישט. מיט א פראמפט-קייט צעטיילט ער עס אויף קלארע טריט: פארשטיין דעם באגריף, אים ערקלערן, שאפן א קאוד-ביישפיל, און צום סוף מאכן א קיצור.

מען הייבט אן שרייבן דעם קאוד. ערשט קומען די געוועלכע imports: google.generativeai as genai, dotenv כדי צו לייענען דעם API שליסל, און os. דערנאך לאדנט מען דעם שליסל און באשטימט דעם מאדעל, אזוי ווי אין אלע פריערדיגע פראגראמען. דערנאך קומט א פונקציע, def chain_prompt(task), וואס פירט אויס די פראמפט-קייט. דער ערשטער טריט איז model.generate_content(f = step1 "Break this task into clear steps: ..."). דא רופט מען אן Gemini, כדי ער זאל אויספירן דעם ערשטן טריט פון דער קייט: צעטיילן די אויפגאבע אויף קלארע טריט.

קייטן, ראוטינג, פאראלעל-ארבעט און אן אינטערוויו-קאוטש (20:15:37)

דער סוף פונעם פראמפט-קייט אגענט (20:15:37)

מען האלט אינמיטן דעם פראמפט-קייט אגענט (prompt chaining). דאס איז אן אגענט וואס צעטיילט א פראגע אין עטלעכע שריט, איינס נאך דאס אנדערע. אינעם ערשטן שריט פרעגט מען Gemini גלייך: אויף וועלכע טיילן קען מען צעטיילן די אויפגאבע פונעם באניצער? דער AI גיט צוריק א צעטיילונג שריט נאך שריט. דערנאך דרוקט מען אויס די צעטיילונג (step breakdown), כדי דער באניצער זאל זען ווי אזוי Gemini האט אויסגעשטעלט די אויפגאבע.

דערנאך קומט שריט צוויי. מען רופט נאכאמאל Gemini דורך model.generate_content, און דער AI שרייבט א דעטאלירטע ערקלערונג וואס איז געבויט אויף דער צעטיילונג פונעם ערשטן שריט. די ערקלערונג דרוקט מען אויך אויס.

אין שריט דריי בעט מען דעם אגענט א לעצטע סיכום. מען רופט ווידער Gemini, און עס קומט צוריק א סיכום וואס א אנפאנגער קען פארשטיין. דער סיכום וועלט אויסגעדרוקט, און די פונקציע גיט צוריק דעם רעזולטאט פון שריט דריי.

צום סוף קומט דער הויפט-טייל פונעם פראגראם. ער קוקט צי מען לאזט לויפן די טעקע גלייך אדער מען אימפארטירט זי. דארט רופט מען די קייט-פונקציע מיט א ביישפיל-פראמפט: "explain Python decorators

with an example". דער קורס זאגט אז מען קען טוישן דעם פראמפט ווי מען וויל, און זען צי דער אגענט גייט דורך אלע שריט.

דער רעזולטאט פון דער קייט (20:17:19)

מען לאזט לויפן דעם פראגראם ווי געווענליך: python prompt_chain_agent.py. מען האט געפרעגט וואס זענען decorators, און מען ערווארט א ערקלערונג אין דריי שריט מיט א סיכום. דער רעזולטאט ווייזט ערשט א צעטיילונג פון דער ערקלערונג אין קלארע שריט:

- שריט איינס: א הקדמה וואס decorators זענען.
- שריט צוויי: וואס מען דארף פריער וויסן (prerequisites), דאס הייסט די יסודות פון וועלכע עס איז געבויט.
- שריט דריי: ווי מען בויט א פשוטן decorator.
- שריט פיר: די קורצע שרייבונג (syntactic sugar).
- שריט פינף: ווי מען באהאנדלט ארגומענטן און צוריקגעגעבענע ווערטן.
- שריט זעקס: ביישפילן פון ווירקליכן לעבן און וואו מען ניצט עס בפועל.
- צום סוף: א סיכום.

מען זעט אז עס קומט טאקע א סיכום אויף דעם גאנצן ענין פון Python decorators. דער קורס באטאנט וויפל אינפארמאציע מען האט באקומען דורכן קייטן-פראצעס.

ראוטינג: דער אגענט קלייבט אליין דעם געצייג (20:18:21)

די נעקסטע שטארקע מעגלעכקייט הייסט ראוטינג (routing). דער אגענט באשליסט אליין וועלכן געצייג אדער מעטאד צו ניצן, לויט דער פראגע פונעם באניצער. דער קורס ערקלערט אז אזוי באשליסט ChatGPT צי צו ניצן דעם בראוזער, Python, זכרון, אדער בילד-פונקציעס. אויך קען מען אזוי ניצן ספעציעלע געצייג. מען בויט א פשוטן ראוטער אין Python, וואס ארבעט לויט וועלכע ווערטער (keywords) עס געפינען זיך אין דער פראגע.

דער קאוד גייט אזוי:

- מען אימפארטירט די געווענליכע זאכן: Google generative AI, dotenv און os.
- דערצו אימפארטירט מען פון tools די פונקציע get_current_time. דאס איז די פונקציע פון א פריערדיגן פראגראם וואס ברענגט די דאטום און צייט.
- מען רופט load_dotenv און שטעלט איין וועלכן מאדעל פון Gemini צו ניצן.
- די פונקציע router_agent באקומט די פראגע (query) און שיקט זי צו דער ריכטיגער לאגיק.
- אויב דאס ווארט time געפינט זיך אין query.lower(), דאס הייסט אויב דער באניצער'ס מעסעדזש רעדט פון צייט, רופט מען א פונקציע וואס ברענגט די ווירקליכע צייט פונעם יעצטיגן מאמענט, און מען גיט עס צוריק.
- אויב דאס ווארט explain געפינט זיך אין דער פראגע, בעט מען Gemini צו שרייבן א פשוטע ערקלערונג, און מען גיט צוריק די ערקלערונג.

- אויב קיין תנאי פאסט נישט, איז דא א "default fallback": response = model.generate_content(query).text. דאס הייסט, מען פרעגט Gemini געוועלך.
- צום סוף גיט מען צוריק די תשובה.

מען רופט די פונקציע מיט צוויי פראגעס: "what is the time now" און "explain recursion".

מען לאזט עס לויפן מיט python router_agent.py. אויף דער ערשטער פראגע ווייזט עס גלייך די יעצטיגע צייט. אויף דער צווייטער ערקלערט עס recursion מיט א משל: אנשטאט טאן די גאנצע ארבעט אליין, פיגורט מען אויס ווי אזוי צו טאן א קליין טייל, און אזוי ווייטער. אין קורצן איז recursion א פונקציע וואס לייזט א פראבלעם דורכדעם וואס זי רופט זיך אליין מיט א פשוטערע ווערסיע פונעם פראבלעם. מען האט באקומען תשובות אויף ביידע פראגעס. דער קורס פירט אויס אז לויט די ווערטער וואס מען ניצט אינעם פראמפט, שאפט דער אגענט אן ענטפער וואס פאסט פאר דעם פראמפט.

פאראלעלע אויספירונג פון אויפגאבעס (20:21:25)

דער נעקסטער ענין איז אויספירן עטלעכע אויפגאבעס אויף איינמאל (parallel execution). Gemini שטיצט צו לויפן עטלעכע פראמפטס אויף איינמאל. די פאראלעלע ארבעט מאכט דעם אגענט שנעלער: אויב דער אגענט דארף טאן פיר אויפגאבעס, טוט ער זיי אלע אין דער זעלבער צייט, אנשטאט איינס נאך דאס אנדערע.

דער קאוד גייט אזוי:

- מען אימפארטירט ווי געוועלך google.generativeai as genai, dotenv און os.
- דאס איינציגע נייע אימפארט איז concurrent.futures. דאס דערלויבט צו לויפן די פראמפטס פאראלעל.
- דערנאך קומט load_dotenv, genai.configure מיטן Gemini API שליסל, און דער נאמען פונעם מאדעל.
- א פונקציע (ask_gemini) שיקט איין פראמפט צו Gemini און גיט צוריק נאר דעם טעקסט.
- א ליסטע פון פיר פראמפטס: "summarize Python in 200 words", "explain AI agents in simple terms", "give a motivational quote", און "write a one-line definition of API". די אויפגאבעס ווערן אויסגעפירט פאראלעל.
- with concurrent.futures.ThreadPoolExecutor() as executor שאפט א "thread pool", כדי צו לויפן עטלעכע אפעראציעס אויף איינמאל.
- results = list(executor.map(ask_gemini, prompts)) פארבינדט יעדן פראמפט מיט דער פונקציע ask_gemini און פירט זיי אויס פאראלעל. אלע רעזולטאטן ווערן אריינגעזאמלט אין א ליסטע.
- א לופ (for r in results) גייט דורך יעדע תשובה פונעם AI און דרוקט זי אויס.

מען לאזט לויפן דעם פראגראם, און אלע פיר פראמפטס ווערן געענטפערט אין עטלעכע סעקונדעס. אנשטאט שרייבן איין פראמפט, ווארטן אויף די תשובה, און דערנאך שרייבן נאך איינס, באקומט מען די תשובות אויף אלע פיר אויף איינמאל.

דער קורס פאסט צוזאם וואס מען האט יעצט צוגעלייגט. דער אגענט קען צעטיילן אן אויפגאבע, אויסקלייבן דעם ריכטיגן געצייג, און אויספירן עטלעכע אויפגאבעס אויף איינמאל. ער ענטפערט אויך אויף א פראפעסיאנעלן און קלוגן אופן. די פארגעשריטענע מעגלעכקייטן ברענגען דעם אגענט פיל נענטער צו AI-געהילפן וואס ארבעטן ביי ריכטיגע ביזנעסער (production level).

דער גרויסער פראיעקט: AI אינטערוויו-קאוטש (20:23:58)

יעצט, מיט אלעם וואס מען האט געלערנט, בויט מען א פראיעקט פון אנהייב ביז סוף: א גאנצער AI אינטערוויו-קאוטש מיט Gemini און Python. דער געהילף שפילט א ריכטיגן אינטערוויוער. ער פרעגט פראגעס, אנאליזירט די תשובות, געדענקט די שוואכקייטן פונעם באניצער, און שאפט א פראפעסיאנעלע, פערזענליכע אפשאצונג (feedback). מען רופט דעם פראיעקט "AI Interview Preparation Coach".

די סטרוקטור פונעם פראיעקט: ווי אנפאנגס, שאפט מען א פאלדער מיט אלע נויטיגע טעקעס:

- .env: היט אפ דעם Gemini API שליסל.
- roles.py: א ליסטע פון פארשידענע פאזיציעס (roles) מיט זייערע שליסל-ווערטער.
- interview_agent.py: די הויפט לאגיק פונעם אגענט.
- memory_manager.py: א לאנג-טערמיניגער זכרון פון די שטארקייטן און שוואכקייטן.
- tools.py: געצייג, ווי צייט, אפשאצונג פון פונקטן (scoring), קאנטראל פון שליסל-ווערטער, א.א.וו.
- router.py: פירט אויס די אויפגאבעס, אנאליזירט פראגעס און שאפט feedback.
- mock_interview.py: לויפט א גאנצע איבונג-אינטערוויו.
- interview_memory.json: היט אפ די היסטאריע פון די אינטערוויוס.

די טעקע roles.py (20:25:31)

מען שאפט די פאלדערס און טעקעס, און לייגט אריין דעם Gemini API שליסל אין דער .env טעקע. די צווייטע טעקע איז roles.py. דארט שרייבט מען א פאר פאכן (domains), כדי דער טשעטבאט זאל אנבאטן אפציעס פאר וועלכע אינטערוויו מען וויל זיך איבן. די ליסטע interview_roles אנטהאלט ניַין פאזיציעס: software engineer, Python developer, data analyst, machine learning, cyber security, cloud engineer, project manager, digital marketer און HR manager. צו יעדער פאזיציע האט מען צוגעלייגט שליסל-ווערטער, ווי למשל OOP, DSA, algorithms, כדי דער AI זאל קענען ברענגען פראגעס לויט די טעמעס.

דערנאך קומט א פונקציע וואס לאזט דעם באניצער אויסקלייבן א פאזיציע:

- זי דרוקט אויס א באגריסונג: "Welcome to AI Interview Coach", און בעט צו קלייבן א פאזיציע מיט וועלכער צו אנהייבן די איבונג-אינטערוויו.
- roles = list(interview_roles.keys()) נעמט אלע פאזיציעס פון דער ליסטע.
- for i, r in enumerate(roles, start=1) גייט דורך אלע פאזיציעס מיט נומערן פון 1 ביז 9, ווייל עס זענען דא ניַין פאזיציעס.
- מיט print(f...) דרוקט מען יעדן נומער מיט דעם טיטל, למשל "9. HR Manager", "8. Digital Marketer".
- choice = int(input(...)) בעט דעם באניצער אריינצושרייבן א נומער. וועו מען שרייבט א נומער פון 1 ביז 9, ברענגט די פראגראם פראגעס פון דעם פאך וואס מען האט אויסגעקליבן.
- role = roles[choice - 1] פארבינדט דעם נומער מיט דער ריכטיגער פאזיציע אין דער ליסטע.
- מען דרוקט צוריק "You selected"..., כדי דער באניצער זאל זען וועלכע פאזיציע ער האט אויסגעקליבן.
- return role, interview_roles[role] גיט צוריק סיי דעם נאמען פון דער פאזיציע, סיי אירע מעטא-דאטא.

די טעקע tools.py: אפשאצן די תשובות (20:27:33)

נאכן אנהייב גייט מען צו tools.py. די ערשטע פונקציע איז keyword_match_score(answer, keywords). זי ציילט וויפל פון די ערוואַרטעטע שליסל-וואָרטער געפינען זיך אין דער תשובה פונעם באניצער. מער שליסל-וואָרטער מיינט א העכערע אפשאצונג:

- answer = answer.lower() פארוואנדלט די תשובה אין קליינע אותיות, כדי די פארגלייכונג זאל נישט אפהענגען פון גרויסע אדער קליינע אותיות.
- matches = sum(1 for k in keywords if k.lower() in answer) ציילט וויפל שליסל-וואָרטער שטימען.
- די פונקציע גיט צוריק די צאל פון די וואָרטער וואס שטימען. יעדע פראגע האט שוין פריער צוגעגרייטע שליסל-וואָרטער. די פראגראם פארגלייכט זיי מיט דער תשובה און גיט אן אפשאצונג.

די צווייטע פונקציע, generate_score(keyword_score, difficulty="medium"), פארוואנדלט די צאל פון שליסל-וואָרטער אין א לעצטע אפשאצונג פון 10. די שווערִיגקייט (difficulty) רוקט א ביסל די אפשאצונג:

- base = keyword_score * 2: יעדעס שליסל-וואָרט לייגט צו צוויי פונקטן.
- אויב difficulty == "hard", וועָרט base -= 1. ביי א שווערער אינטערוויו נעמט מען אראפ א ביסל.
- elif difficulty == "easy", וועָרט base += 1. ביי א גרינגער אינטערוויו גיט מען א קליינעם באנוס.
- כדי די אפשאצונג זאל בלייבן צווישן 1 און 10: return max(1, min(10, base)).

די דריטע פונקציע וויזיט אן וועלכע וויכטיגע פונקטן דער באניצער האט פארפעלט:

- answer = answer.lower(), כדי די פארגלייכונג זאל זיין גלייכמעסיג.
- missing = [k for k in keywords if k.lower() not in answer] שאפט א ליסטע פון די שליסל-וואָרטער וואס זענען נישט געווען אין דער תשובה.
- missing_list = ", ".join(missing), און עס וועָרט צוריקגעגעבן א מעסעדזש: "You missed these important concepts: ... Try including them next time."

דער AI אינטערוויו-קאוטש: זכרון, ראוטער און הויפט-פראגראם (20:30:07)

פארענדיקן tools.py: שטארקייט, שוואכקייט און קורצע פידבעק (20:30:07)

דער קורס וויזיט וויטער דעם קאוד פונעם AI אינטערוויו-קאוטש. די פריערדיגע פונקציע גיט א פידבעק וואס זאגט: "די קאנצעפטן האסטו פארפעלט, פרובירט זיי אריינצונעמען אין די קומענדיגע ענטפערס."

די נעקסטע פונקציע הייסט def classify_strength_or_weakness(score). זי שטעלט אריין דעם באניצערס פערפארמענס אין א קאטעגאריע לויט דער צאל, מיט א פשוטן כלל: אויב דער score איז 7 אדער מער, קומט צוריק "strength" (שטארקייט), אנדערש "weakness" (שוואכקייט). אזוי וועָרט באשטימט צי דער ענטפער איז א שטארקער אדער א שוואכער.

די לעצטע פונקציע אין דער טעקע הייסט def short_feedback. זי גיט א קורצע באמערקונג לויט דער צאל:

- 8 אדער מער: "Excellent answer with strong clarity", א פרעכטיגער ענטפער מיט שטארקער קלארקייט.

• צווישן 6 און 8: "Good answer but can be more detailed", א גוטער ענטפער, נאר ער קען זיין מער דעטאלירט.

• דער מיטעלער שטאפל: "Decent attempt but lacking key concepts", א שיינער פרואוו, נאר עס פעלן הויפט-קאנצעפטן.

• אונטער 4: "Weak answer, need major improvement", א שוואכער ענטפער וואס דארף שטארק פארבעסערט ווערן.

memory_manager.py: א לאנג-טערמיניגער זכרון אין JSON (20:31:09)

די נעקסטע טעקע איז memory_manager.py. דא ווערט אויפגעהיטן א לאנג-טערמיניגער זכרון פון שטארקייטן און שוואכקייטן. עס ווערט באטאנט אז די יסודות זענען וויכטיג: ווייל דאס איז א זכרון-טעקע, דארף מען JSON פארמאט. דעריבער ווערט גלייך אין אנפאנג אימפארטירט json, און ווי געווענליך אויך os)די אפעריירונג-סיסטעם).

מען שטעלט אוועק א זכרון-טעקע מיטן נאמען interview_memory.json. דארט וועלן ווערן אויפגעהיטן אלע אינטערוויו-זכרונות. דער קאוד קאנטראלירט צי די טעקע עקזיסטירט שוין. אויב נישט, שאפט ער א נייע טעקע וואו history, strength און weakness זענען פארלויפיג ליידיג. ווען די שמועסן גייען און דער אינטערוויו פירט זיך, ווערט דער זכרון אנגעפילט אין JSON פארמאט.

די פונקציע save_interview_record(role, question, user_answer, feedback) היט אפ איין פראגע-און-ענטפער רעקארד אינעם לאנג-טערמיניגן זכרון:

• מיט with open(memory_file, "r") as f און json.load(f) = data ווערט דער איצטיגער אינהאלט פון דער טעקע אריינגעלאדן אין א Python dictionary.

• אין דער history ווערט צוגעלייגט א נייער איינטראג מיט) role דער סארט אינטערוויו),) questionדי פראגע וואס דער AI האט געפרעגט),) answerדעם באניצערס ענטפער) און) feedbackדי פולע פידבעק: צאל, שטארקייטן און שוואכקייטן).

• מיט with open(memory_file, "w") as f און json.dump(data, f, indent=4) ווערט דער דערהיינטיגטער זכרון צוריק געשריבן אויפן דיסק, שיין פארמאטירט.

די פונקציע save_strength טוט דאס זעלבע: זי לאדנט דעם זכרון אריין אין א dictionary, לייגט צו די נייע שטארקייט מיט]"strength"data[.append(strength) און שרייבט דאס צוריק אין דער טעקע מיט indent=4. save_weaknesses איז דער זעלבער קאוד, נאר מיט שוואכקייט-אינפארמאציע אנשטאט שטארקייט.

די לעצטע פונקציע, get_memory_summary , נעמט ארויס דעם זכרון פאר פערזענלעכע באריכטן. מיט with open(memory_file, "r") as f און return json.load(f) קומט צוריק דער גאנצער זכרון: history, שטארקייטן און שוואכקייטן.

interview_agent.py: פראגעס און אנאליז מיט Gemini (20:34:19)

די נעקסטע טעקע איז interview_agent.py . דא ווערן אימפארטירט די געווענליכע ביבליאטעקן: google.generativeai, dotenv און os. פון memory_manager ווערן אימפארטירט save_interview_record, save_strength און save_weaknesses, און פון tools אלע געצייג-פונקציעס.

אזוי קען מען דא רופן אלע פונקציעס וואס זענען פריער דעפינירט געווארן. דערנאך קומט load_dotenv פארן API שליסל. די דריי שריט זענען א קביעות אין יעדן פראגראם.

פונקציע generate_question(role), 1, שטעלט צוזאם א פראמפט מיט אן Gemini .f-string שאפט איין אינטערוויו-פראגע, דער קאוד נעמט אראפ איבעריגע ספעיסעס און גיט זי צוריק.

פונקציע 2 אנאליזירט דעם באניצערס ענטפער אויף דער אינטערוויו-פראגע. מען ניצט די ראלע-ספעציפישע דאטא, וואו עס זענען פאראן שווערigקייט-שטאפלען (easy, medium, hard) מיט באזונדערע צאלן. קודם ווערט דער ענטפער געשאצט לויט ווי פיל שליסל-ווערטער ער טרעפט. די פונקציעס פאר דעם זענען שוין פריער געשריבן געווארן, און דא רופט מען זיי נאר.

דערנאך קומט דער פראמפט פאר Gemini: "You are an HR expert and senior interviewer", דו ביסט אן HR מומחה און סניאר אינטערוויוער. Gemini דארף אפשאצן דעם ענטפער פאר דער ראלע וואס דער באניצער האט אויסגעקליבן. עס ווערן אים איבערגעגעבן דעם קאנדידאטס ענטפער, די ענדגילטיגע צאל, און די שטארקייטן און שוואכקייטן לויט די שליסל-ווערטער. Gemini דארף צוריקגעבן א קורצע אנאליז פון צוויי-דריי שורות וואס ווערט געדרוקט נאכן אינטערוויו:

- איין זאץ ווי א רעקרוטער וועט אויפנעמען דעם ענטפער.
- איין פראקטישער עצה ווי זיך צו פארבעסערן.
- די פידבעק זאל זיין נוצלעך אבער נישט שארף.

מיט model.generate_content(analysis_prompt).text באקומט מען די AI פידבעק. דערנאך ווערן די שטארקייטן און שוואכקייטן אפגעהיטן אינעם לאנג-טערמיניגן זכרון וואס איז פריער געשאפן געווארן: שטארקע ענטפערס ווערן געשפייכלט אלס שטארקייטן, שוואכע אלס שוואכקייטן. די פונקציע גיט צוריק: score, improvements, brief_feedback, און ai_feedback פון Gemini.

router.py: דער קלאס InterviewRouter וואס פירט דעם אינטערוויו (20:36:54)

אין router.py ווערן קודם אימפארטירט generate_question און analyze_answer פון interview_agent)פראגע-שאפונג און ענטפער-אנאליז), און save_interview_record און get_memory_summary פון memory_managerאפהיטן רעזולטאטן און לאדן דעם זכרון-סיכום( ).

דער הויפט-קלאס InterviewRouter קאנטראלירט דעם גאנצן לויף פונעם אינטערוויו. די ערשטע פונקציע, init(self, role, role_data), היט אפ די אויסגעקליבענע ראלע (למשל software engineer אדער data science) און די ראלע-דאטא מיט די שליסל-ווערטער. דארט שטייען אויך:

- א ציילער וויפיל פראגעס זענען געפרעגט געווארן.
- די מאקסימום צאל פראגעס.
- א סימן צי מען וואלט אויף אן ענטפער.
- די איצטיגע פראגע.

די פונקציע next_step(self, user_input=None) פירט דעם גאנג:

- אויב self.question_count == 0, ווערט גערופן self.ask_new_question() פאר דער ערשטער פראגע.

- אויב מען ווארט אויף אן ענטפער און עס איז געקומען user_input , ווערט גערופן self.process_answer(user_input).
- דאס חזרט זיך ביז די מאקסימום צאל פראגעס. אין דעם פאל זענען עס דריי פראגעס, דעריבער לויפט עס דריי מאל.
- צום סוף קומט return self.final_report(), וואס גיט צוריק דעם ענדגילטיגן סיכום.

די פונקציע ask_new_question שטעלט generate_question(self.role) = self.current_question, אזוי אז Gemini שאפט די פראגע. זי הייבט אויף דעם ציילער, שטעלט אז מען ווארט אויף דעם באניצערס ענטפער, און גיט צוריק סטרוקטורירטע דאטא מיט type "question" און text מיט דער פראגע. אזוי ווייסט די באניצער-אויבערפלאך (UI) אז דאס איז א פראגע, און ווייסט וואס צו ווייזן.

די פונקציע process_answer(self, user_answer) שטעלט ערשט self.waiting_for_answer = False, ווייל דער ענטפער איז שוין אנגעקומען. דערנאך רופט זי analyze_answer(self.role, user_answer, self.role_data), וואס גיט צוריק די צאל, די פעלנדיגע שליסל-ווערטער און די AI פידבעק. מיט save_interview_record ווערן role, current_question, user_answer און feedback אפגעהיטן אינעם לאנג-טערמיניגן זכרון. צוריק קומט type "analysis" מיט: א צאל פון 1 ביז 10, א קורצע פידבעק, פארבעסערונג-עצות (ווי פעלנדיגע שליסל-ווערטער), און א דעטאלירטע קריטיק פון Gemini.

די לעצטע פונקציע, final_report, רופט get_memory_summary און גיט צוריק type "final_report" מיט: די צאל פראגעס, די לעצטע פינף שטארקייטן, די לעצטע פינף שוואכקייטן, און די מעלדונג "Interview completed! Review your performance summary below". דאס באצייכנט דעם סוף פונעם אינטערוויו.

mock_interview.py: דער הויפט-לופ און לויפן דעם קאוד (20:41:28)

די לעצטע טעקע איז mock_interview.py. פון roles ווערט אימפארטירט choose_role, וואס ווייזט די פאראנענע ראלעס און גיט צוריק דעם באניצער אויסוואל. פון router ווערט אימפארטירט InterviewRouter, דער קאנטראלער פונעם גאנצן פראגע-און-ענטפער לויף.

די פונקציע run_interview דרוקט א טיטל "AI Interview Coach" מיט א שורה פון = צייכנס. דערנאך:

- choose_role() = role, role_data: דער באניצער קלייבט אויס די ראלע.
- InterviewRouter(role, role_data) = router: דער פירער פונעם אינטערוויו ווערט איניציאליזירט.
- עס ווערט געדרוקט "Starting the interview".

דערנאך קומט א while True לופ וואס פרעגט ווייטער פראגעס ביזן סוף:

- router.next_step() = step.
- פאל 1: אויב step["type"] == "question", דרוקט מען די פראגע מיט איר נומער און דעם טעקסט, און נעמט דעם ענטפער מיט input("Your answer "). דערנאך גייט דער ענטפער דורכן ראוטער מיט router.next_step(answer) = analysis.
- מען דרוקט די פידבעק: די צאל, די קורצע פידבעק, די פעלנדיגע פונקטן און די AI פידבעק.
- פאל (elif) 2: אויב step["type"] == "final_report", דרוקט מען "Final Interview Report", די צאל פראגעס וואס זענען געענטפערט געווארן, און די שטארקייטן. אויב עס זענען נאך נישטא קיין שטארקייטן,

דרוקט מען "No strengths recorded yet". אזוי אויך פאר די שוואכקייטן. צום סוף ווערט געדרוקט די שלוס-מעלדונג מיט א שורה = צייכנס, און break גייט אריס פונעם לופ.

צום סוף שטייט א בלאק וואס רופט run_interview וועלכע ווערט דירעקט אויסגעפירט.

כדי צו לויפן דעם קאוד קאפירט מען דעם פאלדער-לאקאציע און מען לייגט אים אריין אין cmd, כדי צו נאוויגירן צום פאלדער. דערנאך לויפט מען python mock_interview.py, און מען זעט ווי דער אויסדרוק הייבט זיך אן צו ווייזן.

געזעראטיוו AI אגענטן: יסודות, ארבעט און ארכיטעקטור (20:44:32)

א דעמא: דער AI אינטערוויו-קאוטש (20:44:32)

דער אפשניט הייבט זיך אן מיט א דעמא פון אן אפליקאציע וואס הייסט "AI אינטערוויו-קאוטש". וועלכע מען עפנט זי, באקומט מען א באגריסונג: ברוכים הבאים צום AI אינטערוויו-קאוטש, קלייבט אויס א ראלע און הייבט אן אייערע פראבע-אינטערוויוס.

מען קען אויסקלייבן פון פיר ראלעס: software engineer, Python developer, data analyst און machine learning engineer. דער קורס באמערקט אז מען קען צולייגן נאך ראלעס לויט די אייגענע ידיעות.

אין דער דעמא קלייבט מען אויס נומער דריי, data analyst . די אפליקאציע זאגט אז מען האט אויסגעקליבן data analyst, און דער אינטערוויו הייבט זיך אן. די ערשטע פראגע: שטעלט אייך פאר אז מען גיט אייך אן אויפגאבע אויסצופארשן א פלוצלונגדיקע, אומדערקלערטע אראפפאל אין א וויכטיגן געשעפט-מאס (metric). פירט מיך דורך דעם פראצעס וויאזוי איר וואלט געפונען די ווארצל-סיבה און צוגעגרייט אייערע רעזולטאטן.

דער קורס ערקלערט אז דאס איז א פראקטישע פראגע, און מען דארף ענטפערן אזוי ווי אין אן אמת'ן אינטערוויו: וועלכע שריט מען וואלט געטון.

דער ענטפער וואס ווערט אריינגעשריבן אין דער דעמא:

- איך וואלט אנגעהויבן מיט באשטעטיגן אז די דאטא איז ריכטיג.
- דערנאך וואלט איך שנעל באשטימט צי דער אראפפאל קומט פון אינערליכע ענדערונגען אדער פון דרויסנדיגע פאקטארן, און זיך קאנצענטרירט נאר אויף די געגנטער וואו מען זעט אומגעווייניטלעכע מוסטערן.
- נאכדעם וואס איך האב אפגעזונדערט די מערסט מעגליכע סיבה, וואלט איך צוגעגרייט א קורצן איבערבליק פון די אויפטועכטסן און עצות פאר דער מאנשאפט.

נאכן אריינשיקן דעם ענטפער באקומט מען פידבעק. די צייכן (score) איז 1 פון 10. די קורצע אפשאצונג: א שוואכער ענטפער, דארף א סך פארבעסערונג. אונטער "וואס פעלט" שטייט: מען האט אויסגעלאזט וויכטיגע באגריפן ווי SQL, Excel, PowerBI, Pandas און וויזואליזאציע, און מען זאל זיי אריינענעמען דאס קומענדיגע מאל. דערנאך קומט אן AI-פידבעק מיט אן אפשאצונג פונעם קאנדידאט'ס ענטפער.

מען קען ווייטער גיין צו דער צווייטער פראגע: פירט מיך דורך אייער געוועזענליכן פראצעס וועלכע איר באקומט א נייע אנאליטישע בקשה פון א געשעפט-פארטרעטער (stakeholder), פון אנהייב, דורך פארשטיין, ביז פארלייגן די רעזולטאטן.

דער קורס זאגט אז די גאנצע פראגראם קען העלפן ביים צוגרייטן זיך צו אינטערוויוס. עס איז א טשאט-באט וואס מען האט געבויט מיט Gemini.

דער סדר פון דער לעקציע (20:46:05)

דערנאך ווערט פארגעלייגט דער סדר פון דער לעקציע:

- א קורצע הקדמה צו געניראטיווע AI אגענטן.
- די הויפט-באשטאנדטיילן פון Gen AI אגענטן.
- וויאזוי געניראטיווע AI אגענטן ארבעטן.
- ארכיטעקטורן און אינסטאלירונג (deployment).
- פארשטארקן AI אגענטן מיט דרויסנדיגע געצייג.
- שווערigקייטן און באגרעניצונגען פון Gen AI אגענטן.
- צוקונפטיגע טרענדן און פארשריטן אין AI אגענטן.

וואס איז א געניראטיווער AI אגענט? (20:46:35)

דער קורס ערקלערט אז אן AI אגענט איז אין גרונט אן LLM וואס קען טון אקציעס, אליין, אן מענטשליכער השגחה, און דערצו אויך טראכטן (reason) וועגן די אויפגאבעס. מער פארמעל: אן AI אגענט איז א וייכווארג-גוף וואס ארבעט אליין מיט זיין סביבה, נעמט באשלוסן און פירט אויס אויפגאבעס. געניראטיווע AI אגענטן גייען נאך א שריט ווייטער: זיי באנוצן זיך מיט גרויסע שפראך-מאדעלן ווי די GPT -מאדעלן פון OpenAI כדי צו שאפן קלוגע ענטפערס און טון אויפגאבעס וואס גייען ווייטער פון אויטאמאציע געבויט אויף פעסטע כללים.

די אנטוויקלונג פון AI אגענטן (20:47:39)

עס ווערן באשריבן דריי שטאפלען:

- טראדיציאנעלע אגענטן געבויט אויף כללים: זיי זענען באגרעניצט צו פריער באשטימטע כללים. גיט מען זיי א פראצעס מיט שריט, טוען זיי עס אויס און ענדיגן, אבער זיי גייען נישט קיין שריט ווייטער.
- אגענטן געבויט אויף מאשין-לערנען: זיי קענען לערנען פון דאטא, אבער זיי דארפן סטרוקטורירטע אינפוט.
- געניראטיווע AI אגענטן: זיי קענען פארשטיין, טראכטן און שאפן מענטשנגלייכע ענטפערס אליין. זיי פארשטייען די סביבה, לערנען פון איר, טוען שריט לויט די פרינציפן אויף וועלכע מען האט זיי טריינירט, און זאגן אויך די סיבה פארוואס זיי האבן געטון א געוויסן שריט.

פארוואס זיי זענען וויכטיג (20:48:10)

דער קורס ברענגט דריי הויפט-פונקטן:

- עפעקטיווקייט: זיי אויטאמאטיזירן איבערגעחזר'טע ארבעט מיט ווייניג מענטשליכער אריינמישונג.
- הויכע סקאלירבארקייט: זיי באהאנדלען גרויסע מאסן ארבעט, למשל קונה-שטיצע, שאפן אינהאלט און נאך.
- אינטעליגענץ: זיי קענען זיך צופאסן דינאמיש צו פארשידענע באנוצן, און קיינער דארף נישט שרייבן כללים פאר זיי.

OpenAI'ס אגענט Operator (20:48:40)

דערנאך קאנצענטרירט זיך די לעקציע אויף OpenAI, וואס האט ארויסגעלאזט צוויי נייע אגענטן: Operator און Deep Research. לויטן קורס זענען די צוויי גאר גרויסע נייעס פאר אלע וואס אנטוויקלען AI אגענטן.

Operator איז דער ערשטער AI אגענט פון OpenAI. ער שפילט נאך מענטשליכע אקציעס אינעם בראוזער: ער קען סקראלן, שרייבן, קליקן און נאוויגירן וועב-זייטלעך פאר אייך, פונקט ווי א מענטש. דער קורס באטאנט א פונקט וואס פילע האבן נישט באמערקט: דער אגענט באזירט זיך אויף א גאנץ נייעם מאדעל וואס הייסט CUA, Computer Using Agent. אנדערש ווי די געווענליכע GPT-מאדעלן, וואס מען שרייבט זיי א פראמפט און באקומט צוריק טעקסט, איז דער מאדעל טריינירט ארויסצוגעבן מויז- און קלאוויאטור-קליקס. דאס מאכט אים אזוי שטארק אינעם בראוזער.

דערצו טראכט ער וועגן יעדער אקציע. לויטן קורס איז דאס א זאך וואס מען האט פריער נישט געזען: ער טוט נישט סתם אקציעס, נאר ער באטראכט יעדן שריט און וואס ער זאל טון ווייטער. פונקט ווי אין דער אלגעמיינער באשרייבונג פון געניראטיוע אגענטן, נעמט Operator א שריט לויט דער סביבה, גיט די סיבה פארוואס ער האט עס געטון, און פרעגט אייך וואס צו טון ווייטער.

דאס ביישפיל: אויב אירווילט באשטעלן א בילעט, גייט ער איין אין אלע וועב-זייטלעך, זוכט בעסערע פלי-אפציעס: די ביליגסטע, די מיט ווייניגער שעהען פלי און די בעסטע צייטן. צום סוף קלייבט ער אויס דאס בעסטע לויט אייערע וואונטשן, און פרעגט צי ער זאל גיין ווייטער מיטן באשטעלן.

די באגרעניצונגען פון Operator (20:50:42)

דער קורס דערמאנט צוויי באגרעניצונגען:

- געניקייט: זי איז נאך נישט גאנץ הויך. אבער אין פארגלייך מיט פריערדיגע פראיעקטן, וואו מען האט באפוילן GPT-מאדעלן צו טון אקציעס אינעם בראוזער מיט די בילד-פעאיגקייטן פון GPT-4 (vision), איז זי גאר הויך. יענע פריערדיגע לייזונגען האבן שטארק געהאט שוועריגקייטן ווען א פאפ-אפ פענצטער האט זיך פלוצלינג באויזן אויף א וועב-זייטל. דער אגענט איז צומישט געווארן, האט עס נישט געקענט צומאכן, און דער באניצער האט געמוזט אליין אריינשפרינגען און עס צומאכן.
- הויכע קאסטן: דערווייל איז Operator בנמצא נאר אויפן פלאן פון $200 און העכער, אז מען קען אננעמען אז ער וועט זיין גאנץ טייער אויך דורכן API. אבער דער קורס האלט אז אויב מען קאנצענטרירט זיך אויף די ריכטיגע זאכן, וועלן די קאסטן נישט זיין קיין פראבלעם.

OpenAI'ס אגענט Deep Research (20:51:48)

דער צווייטער אגענט פון OpenAI הייסט Deep Research. ער איז טריינירט צו מאכן גרינטליכע פארשונג: ער זוכט אויפן וועב, ציט ארויס די נויטיגע קוואלן, און שטעלט צונויף די אינפארמאציע אין א פולשטענדיג דאקומענט אדער באריכט. די נייע זאך איז אז ער ווערט געטריבן פונעם נייעם מאדעל o3. ווי Operator, גיט ער נישט סתם צוריק רעזולטאטן פון זוכן, נאר ער טראכט וועגן יעדן קוואל און וועלכע אינפארמאציע ער דארף נאך זוכן. לויטן קורס גיט Deep Research נישט סתם ארויס דאס דורכשניטליכע וואס געפינט זיך אויפן אינטערנעט, ווי געווענליכע GPT-מאדעלן, נאר ער שטעלט צונויף נייע אויפטועכטסן.

אנווענדונגען אין דער וירקליכקייט (20:52:49)

- אגענטן פאר קונה-שטיצע: אויטאמאטיזירן ענטפערס פון א הילף-דעסק.

- שטארקע AI אסיסטענטן, ווי "co-pilots" פאר פראפעסיאנאלן.
- פארשונג- און אנאליז-אגענטן: צווזאמפאסן דאטא, איבערבליקן פון ליטעראטור און נאך.
- קרעאטיוו שרייבן און שאפן אינהאלט: אויטאמאטיזירן פאסטס אויף סאציאלע מעדיע, אימעילס א.א.וו.
- AI-קאוד-אסיסטענטן ווי GitHub Copilot, OpenAI Codex און נאך.

די הויפט-באשטאנדטיילן (20:53:21)

- יסוד-מאדעלן (foundation models): GPT-4 פאר נאטירליכע שפראך-באהאנדלונג, E·DALL פאר שאפן בילדער, און Whisper פאר איבערמאכן רייד אויף טעקסט.
- פראמפט-אינזשעניריע און באהאנדלונג פון קאנטעקסט: אויסשטעלן עפעקטיוע פראמפטן כדי דער אגענט זאל געבן פאסיגע ענטפערס; באנוצן few-shot און zero-shot לערנען פאר צופאסיגקייט; און אגענטן וואס געדענקען פריערדיגע שמועסן מיטן באניצער.
- זכרון און לאנג-טערמיניגע קאנטעקסט: AI אגענטן באהאנדלען געוועלך קורצע פראמפטן, אבער מעטאדן מיט זכרון, למשל דער memory-מאדול פון LangChain, מאכן מעגליך לאנג-טערמיניג טראכטן. אויפהיטן און ארויסנעמען דאטא וואס איז ספעציפיש פארן באניצער, פאר פערזענליכע ענטפערס, איז אויך א הויפט-ציל.
- געצייג און API-פארבינדונגען: באנוצן דרויסנדיגע APIs פאר דאטא אין לייוו-צייט, למשל וועטער, אקציע-פרייזן, נייעס, און פארבינדן דאטאבאזעס פאר ארויסנעמען סטרוקטורירטע אינפארמאציע.
- אליינשטענדיג באשליסן: דער אגענט באטראכט עטליכע אפציעס איידער ער באשליסט, און פלאנירונג און טראכט-מעטאדן פארבעסערן די ענטפערס.

וויאזוי געווארטיוועט אגענטן ארבעטן (20:54:52)

ערשטער שריט, באהאנדלונג פון אינפוט און באוואוסטזיין פון קאנטעקסט: דער אגענט אנאליזירט וואס דער באניצער שרייבט, דערקענט די כוונה און ציט אויס די וויכטיגע פרטים. אין א רייזע-באשטעל אגענט, למשל, דערקענט ער פון דער פראגע דעם ציל-ארט, דעם דאטום און דעם בודזשעט. דער קורס פארבינדט עס מיטן פריערדיגן ביישפיל: מען גיט פונקט A און פונקט B, די בעסטע צייט צו פליען, וויפיל שעהען מען ווייל פליען און דעם ציל. דער אגענט שאצט אפ אלעס אין לייוו-צייט: דעם פרייז, די שעהען, די פלי-אפציעס, די באקוועמליכקייטן אויפן עראפלאן און דעם בודזשעט, און לויט דעם באשליסט ער.

צווייטער שריט, שאפן קלוגע ענטפערס:

- RAG, retrieval augmented generation: דער AI זוכט אין דרויסנדיגע דאטאבאזעס איידער ער ענטפערט.
- Chain of Thought (CoT) טראכטן: דער AI צעטיילט פראבלעמען אין קליינע שריט.
- fine-tuning פון מאדעלן: AI ספעציעל טריינירט פאר געוויסע אינדוסטריעס. בויט מען א אגענט פאר א באשטימטן פעלד, ווי געזונט, געזעץ אדער קונה-שטיצע, טוט מען אים fine-tune לויט דעם.

דריטער שריט, באשלוס-ביימער און טראכטן אין עטליכע שריט: דער AI קען נאכמאכן א טראכט-פראצעס און צעטיילן קאמפליצירטע פראגעס אין שריט וואס מען קען באהאנדלען. דאס ביישפיל: וועט מען עפנט DeepSeek, אדער די פרעמיום ווערסיע פון ChatGPT, איז דא אן אפציע "think". דריקט מען דערויף, ערקלערט ער דעם טראכט-פראצעס וואס ער גייט דורך און וויאזוי ער פלאנירט צו ברענגען זיין ענטפער.

פערטער שריט, באהאנדלען אומזיכערקייט און פארריכטן טעותים: דער AI פארריכט זיך אליין דורך א צייכן פון זיכערקייט (confidence scoring) און דורך פידבעק-קרייזן פון מענטשן.

צום לעצט, fine-tuning און reinforcement learning, באקאנט ווי RLHF: טריינירן מיט א מענטש אינעם קרייז כדי צו פארבעסערן די ענטפערס. דער AI לערנט פון אמת'ע שמועסן און פאסט זיך צו מיט דער צייט. דער קורס רופט עס לערנען איבער צייט: ער פאסט זיך צו צו דער ווירקליכער סביבה און ברענגט די בעסטע מעגליכע לייזונג.

ארכיטעקטור און אינסטאלירונג (20:57:26)

- אליינשטייענדיגע קעגן מולטי-אגענט סיסטעמען: אן אליינשטייענדיגער אגענט טוט אויפגאבעס אליין, למשל ChatGPT. אין א מולטי-אגענט סיסטעם ארבעטן עטליכע אגענטן צוזאמען אויף א קאמפליצירטער אויפגאבע, למשל איינער פארן צוזאמפאסן און א צווייטער פארן אויספירן.
- אן אגענט-ארקעסטראטאר: א צענטראלער אגענט קאארדינירט עטליכע אונטער-אגענטן. אין א פארשונג-אסיסטענט, למשל, זאמלט דער ערשטער אגענט די נויטיגע דאטא, דער צווייטער פאסט צוזאמען די אינפארמאציע, און דער דריטער פארמאטירט דעם רעזולטאט.
- אגענטן אין דער כמארע (cloud) קעגן אגענטן אויפן עדזש (edge): די כמארע-אגענטן רעכענען אויף סערווערס, למשל OpenAI API, Azure AI. עדזש-אגענטן לויפן אויף לאקאלע מכשירים פאר שנעלערער באהאנדלונג, למשל Tesla'ס AI אין זעלבסט-פארנדיגע קארס.
- באנוצן APIs צו בויען אגענטן: דער API פון OpenAI פאר GPT-4 געטריבענע אגענטן, דער LangChain פריימווארק פאר ארקעסטרירן אגענטן, און LlamaIndex פאר אגענטן וואס ארבעטן מיט דאקומענטן.

פארשטארקן אגענטן מיט דרויסנדיגע געצייג (20:58:28)

דער לעצטער טייל פונעם אפשניט הייבט אן דעם קומענדיגן שטאפל: פארשטארקן AI אגענטן מיט דרויסנדיגע געצייג. דער ערשטער שריט איז פארבינדן מיט דאטאבאזעס: AI אגענטן קענען אנפרעגן SQL און NoSQL דאטאבאזעס כדי ארויסצונעמען סטרוקטורירטע דאטא.

אגענטיק AI: יסודות, באגרענעצונגען און צוקונפט (20:59:00)

וועב-סקרעיפערס, וועקטאר-דאטאבאזעס און אויטאנאמע אגענטן (20:59:00)

דער קורס פירט ווייטער מיט דעם ענין פון פארבינדן AI-אגענטן מיט וועב-סקרעיפערס און API'ס. עס ווערט ערקלערט אז AI-אגענטן קענען ברענגען דאטא אין רעאלער צייט (realtime) דורך כלים ווי BeautifulSoup, Scrapy און OpenAI plugins.

דערנאך קומט דער ענין פון וועקטאר-דאטאבאזעס פאר צוריקדערמאנען דעם קאנטעקסט. כלים ווי FAISS און ChromaDB היטן אפ א לאנג-טערמיניגן זכרון פאר AI-אגענטן. דאס ביישפיל וואס ווערט געברענגט: לעגאלע AI-אגענטן, וואס ארבעטן אין געזעץ-פעלד, ציען ארויס פריערדיגע געריכט-פאלן (case law) פון וועקטאר-embeddings.

צום לעצט ווערט דערמאנט דער ענין פון אויטאנאמע אגענטן און ארבעטס-פלוסן (workflows). די בעסטע ביישפילן דערפאר זענען AutoGPT און BabyAGI, אגענטן וואס פארבעסערן זיך אליין דורך לערנען פון

פריערדיגע אויפגאבעס. דערצו ווערט דערמאנט א סיסטעם וואס איז א סעט פון עטלעכע AI-אגענטן וואס ארבעטן צוזאמען כדי צו לייזן א קאמפליצירטע פראבלעם.

שווערigקייטן און באגרענעצונגען פון געגנעראטיווע AI-אגענטן (20:59:45)

דאס איז, לויט דעם קורס, דער הויפט-טייל: די שווערigקייטן און באגרענעצונגען. עס ווערן אויסגערעכנט פיר הויפט-פונקטן:

- עטישע באטראכטונגען: די ערשטע זארג איז AI-פארטייאישקייט (bias) און יושר (fairness). דערצו דארף מען זיך אפגעבן מיט AI-"האלוצינאציעס", דאס הייסט ווען דער מאדעל שאפט פאלשע אינפארמאציע.
- קאמפיוטער-קאסטן און עפעקטיווקייט: דאס איז א גרויסע זארג אין אלע אינדוסטריעס. גרויסע AI-מאדעלן פאדערן א סך קאמפיוטער-כח. עס ווערט אויך דערמאנט אז עס איז טייער צו האלטן אסאך טריינירטע מאדעלן אדער צו דינגען מענטשלעכע פראגראמירער, און מען דארף סטראטעגיעס צו מאכן די AI-אגענטן עפעקטיווער.
- זיכערהייט-ריזיקעס און דאטא-פריוואטקייט: עס איז דא א ריזיקע פון דאטא-ליקס און דאטא-איינברוכן. AI "jailbreaking", דאס הייסט אויסטריקן דעם מאדעל ארויס פון זיינע כללים, ווערט באצייכנט אלס איינע פון די העכסטע זיכערהייט-סכנות. די הויפט-לייזונג וואס מען דארף זיך אויף קאנצענטרירן איז אויפשטעלן א זיכערע AI-ארכיטעקטור.
- לעגאלע ענינים און "קאמפלייענס": דאס איז בעיקר א זארג אויף דער שטאפל פון גרויסע פירמעס (enterprise). מען מוז זיך האלטן צו GDPR, HIPAA און AI-רעגולאציעס. אויך איז וויכטיג די שאלה פון אחריות ווען AI נעמט באשלוסן.

צוקונפטיגע טרענדס ביי AI-אגענטן (21:00:45)

דער לעצטער שטאפל פון דעם טייל באהאנדלט די צוקונפטיגע טרענדס און פארשריטן:

- AI-אגענטן מיט געפילישער אינטעליגענץ, אויך גערופן affective AI: AI וואס קען דערקענען געפילן און צופאסן איר ענטפער. דער ביישפיל: א טעראפעטישער AI-chatbot פאר גייסטיגע געזונט.
- סיסטעמען וואס פארבעסערן זיך אליין: ביישפילן זענען meta-learning און AutoML, AI-אגענטן וואס לערנען אליין פון נייע דאטא. ביישפיל: AI וואס פאסט צו מארקעטינג-קאמפיינס לויט ווי די קונים רעאגירן.
- AGI און דער וועג צו "סופער-אינטעליגענץ": דאס ענדיגט זיך שטענדיג מיט דער פראגע צי AI-אגענטן וועלן זיך אנטוויקלען צו AGI (artificial general intelligence), און צי די פאלגן פון AI וואס לערנט זיך אליין זענען רעאל. דאס, ווערט געזאגט, איז די הויפט-פראגע.
- די ראלע פון AI-אגענטן אין דער צוקונפט פון ארבעט: מען זעט שוין אסאך אגענטן ווי Copilot וואס העלפן מענטשן אין טאג-טעגליכע אויפגאבעס, צי אין ביזנעס, געזונט-ווען אדער חינוך. אויך גייט פאר אסאך אויטאמאציע און טראנספארמאציע אין דער IT-אינדוסטריע, וואס וועט פארמינערן די ארבעטס-שעהן און דעם דרוק אויף די אנגעשטעלטע, דורך אויטאמאטיזירן די לאנגווייליגע, רוטינע אויפגאבעס.

קאפיטל איינס: וואס איז אגענטיק AI? (21:02:00)

דא הייבט זיך אן א נייער טייל פונעם קורס. אין דעם ערשטן קאפיטל בויט מען א קלארן יסוד: וואס אגענטיק AI איז, ווי עס האט זיך אנטוויקלט, און פארוואס עס ווערט איינס פון די וויכטיגסטע פארשריטן אין AI היינט.

די הגדרה: אגענטיק AI באדייט AI-סיסטעמען וואס קענען אליין האנדלען כדי צו דערגרייכן א ציל, אנשטאט נאר צו ענטפערן אויף איינצלנע אנווייזונגען. ביי טראדיציאנעלע AI-סיסטעמען איז דער פארקער גאנץ פשוט: מען גיט אריין עפעס און די סיסטעם גיט ארויס א רעזולטאט. דאס הייסט א "request-response" מאדעל. אגענטיק AI ברענגט אריין התנהגות וואס ווערט געטריבן פון א ציל, די מעגלעכקייט צו נעמען באשלוסן, די באנוץ פון דרויסנדיגע כלים, און איבערחזרנדיגע (iterative) טראכטן, וואו די סיסטעם קען אפשאצן און פארבעסערן אירע אייגענע רעזולטאטן.

א פשוטער אופן עס צו פארשטיין: גרויסע שפראך-מאדעלן (LLM) שאפן אינהאלט; אגענטיק AI טוט אויס אויפגאבעס. אגענטיק סיסטעמען פארבינדן שפראך-מאדעלן מיט פלאנירן, זכרון און כלים, אזוי אז זיי קענען דורכפירן קאמפליצירטע ארבעטס-פלוסן מיט ווייניג מענטשלעכע איינמישונג. דער ביישפיל: אנשטאט צו בעטן א AI צו שרייבן אן אימעיל, קען אן AI-אגענט אנאליזירן די סיטואציע, צונויפזאמלען די נויטיגע אינפארמאציע, צוגרייטן א בריוו, און אפילו אים אליין אוועקשיקן. דאס איז אן איבערגאנג פון פאסיווע אינטעליגענץ צו אינטעליגענץ וואס טוט.

דריי שטאפלען פון אנטוויקלונג (21:03:30)

די אנטוויקלונג צו אגענטיק AI איז געשען אין דריי הויפט-שטאפלען:

- ערשטער שטאפל, טראדיציאנעלע AI-סיסטעמען: מערסטנס געבויט אויף כללים (rule-based) אדער טריינירט פאר זייער ספעציפישע אויפגאבעס, ווי בילדער-קלאסיפיקאציע, דערקענען שווינדל, אדער רעקאמענדאציע-מאשינען. זיי זענען געווען שטארק אבער באגרענעצט, ווייל זיי האבן געדארפט פארויס-באשטימטע לאגיק און האבן זיך נישט גרינג געקענט צופאסן.
- צווייטער שטאפל, גרויסע שפראך-מאדעלן (LLM): זיי האבן געברענגט גרויסע פארבעסערונגען אין טראכטן, שמועסן און שאפן אינהאלט, און האבן ערלויבט AI צו רעדן מיט מענטשן אויף א סך נאטירלעכערן אופן. אבער זיי האבן נאך געהאט באגרענעצונגען: נישט קיין דויערדיגן זכרון, נישט קיין מעגלעכקייט אליין דורכצופירן באשלוסן, און א באגרענעצטן פארקער מיט דרויסנדיגע כלים.
- דריטער שטאפל, דער וויכטיגסטער, אגענטיק AI: שפראך-מאדעלן ווערן פארבונדן מיט פלאן-סיסטעמען, זכרון-מעכאניזמען, כלים, און דורכפיר-שלייפן (execution loops). דאס ערלויבט די סיסטעם צו ארבעטן אנגייענדיג צו א באשטימטן ציל, ענליך ווי א מענטשלעכער געהילף ארבעט.

פארוואס אגענטיק AI איז וויכטיג, און אירע הויפט-אייגנשאפטן (21:04:50)

עס ווערט ערקלערט אז אגענטיק AI איז וויכטיג וויל עס מאכט מעגלעך אויטאמאציע פון טראכטן, נישט נאר פון גשמיותדיגער אדער רוטינער ארבעט. דריי הויפט-געביטן:

- אויטאמאציע פון טראכטן: אגענטן קענען צעטיילן א פראבלעם אין טריט, אפשאצן רעזולטאטן און ענדערן דעם צוגאנג אן שטענדיגע מענטשלעכע השגחה.
- אויטאמאציע פון כלים: אגענטן ארבעטן מיט API'ס, דאטאבאזעס, ווייכווארג-פלאטפארמעס און דרויסנדיגע אפליקאציעס. דאס מאכט פון AI, אנשטאט א פשוטן שמועס-פענסטער, א סיסטעם וואס טוט טאקע אויס פעולות.
- אויטאמאציע פון אויפגאבעס: קאמפליצירטע ארבעטס-פלוסן ווי מארקעט-פארשונג, שאפן באריכטן, דאטא-אנאליז, קונה-סערוויס, אדער אפילו הילף ביים אנטוויקלען ווייכווארג, קענען היינט ווערן טיילווייז אדער אינגאנצן אויטאמאטיזירט.

פון א ביזנעס- און עקאנאמישן שטאנדפונקט, האלטן אסאך מומחים אז אגענטיק AI איז דער קומענדיגער גרויסער שפרונג אין פראדוקטיווקייט, נאך cloud computing און מאביל-טעכנאלאגיע.

מערסטע אגענטיק לייזונגען האבן פינף באשטאנדטיילן: (1) באשטימען דעם ציל, פארשטיין וואס מען דארף דערגרייכן; (2) פלאנירן, באשליסן ווי אזוי דאס צו דערגרייכן; (3) דורכפירן, אויספירן פעולות מיט כלים אדער סיסטעמען; (4) זכרון, אפהיטן אינפארמאציע פון פריערדיגע טריט; (5) א feedback-שלייף, אפשאצן רעזולטאטן און פארבעסערן. עס ווערט באמערקט אז דער סטרוקטור איז זייער ענליך צו ווי מענטשן לייזן פראבלעמען אין לעבן.

רעאלע ביישפילן פון פריע איינפירונג: אויטאנאמע פארשונגס-געהילפן, AI-כלים פאר קאוד וואס קענען געפינען פעלערן און ארויסלאזן אפליקאציעס, פערזענליכע פראדוקטיווקייט-געהילפן, ביזנעס-אויטאמאציע-פלאטפארמעס מיט אסאך אגענטן, און סיסטעמען וואס ארגאניזירן ארבעטס-פלוסן. פירמעס עקספערימענטירן אקטיוו דערמיט כדי צו פארבעסערן עפעקטיווקייט און פארמינערן קאסטן.

קאפיטל צוויי: טראדיציאנעלע AI, LLM און אגענטיק AI (21:07:00)

אין דעם קאפיטל גייט מען טיפער אין די יסודות. עס ווערט באטאנט אז איידער מען בויט אגענטן, מוז מען פארשטיין ווי זיי טראכטן, ווי זיי ארבעטן און וואס מאכט זיי אויטאנאם.

טראדיציאנעלע AI איז געבויט געווען פאר ספעציפישע אויפגאבעס, ווי spam-דערקענונג, רעקאמענדאציע-מאשינען אדער בילדער-דערקענונג. זיי ארבעטן גוט אין א באגרענעצטן ראם, אבער זיי טראכטן נישט ווייטער פון וואס זיי זענען טריינירט, נעמען נישט קיין זעלבסטשטענדיגע באשלוסן, און פאסן זיך נישט צו ווען דער קאנטעקסט ענדערט זיך. קורץ: שמאל און ספעציפיש.

LLM'ס זענען געווען א גרויסער דורכברוך: זיי פארשטייען נאטירלעכע שפראך, שאפן אינהאלט, ענטפערן קאמפליצירטע פראגעס און קענען אפילו נאכמאכן טראכטן. זיי זענען טריינירט אויף ריזיגע דאטא-סעטס דורך פירמעס ווי OpenAI, Google DeepMind און Meta. אבער זיי ארבעטן נאך אלץ אין א פשוטן פלוס: פראמפט, ענטפער, סוף. זיי פירן נישט אליין אויס אויפגאבעס און פירן נישט קיין צילן אויף לאנגע צייט. דעריבער זענען זיי, כאטש קלוג, נאך אלץ רעאקטיווע סיסטעמען.

אגענטיק AI פארברייטערט LLM'ס צו סיסטעמען געטריבן פון א ציל: פלאנירן פעולות, באנוצן כלים, אפהיטן זכרון, אפשאצן רעזולטאטן און זיך שטענדיג פארבעסערן. דער פלוס ענדערט זיך אינגאנצן: פראמפט, ענטפער, פלאן, פעולה, אָבסערווירן, פארבעסערן, און ווידער פון אנפאנג. דער שלייף שאפט די אויטאנאמיע.

אויטאנאמיע, פלאנירן און זכרון (21:09:30)

אויטאנאמיע: סיסטעמען געבויט אויף פראמפטס זענען אינגאנצן אפהענגיג פון מענטשלעכע אנווייזונגען, למשל "שרייב מיר א באריכט" אדער "פאסס מיר צוואמען דעם דאקומענט". די סיסטעם ענדיגט און שטעלט זיך אפ; קיין איניציאטיוו, קיין המשך, קיין זעלבסטשטענדיג טראכטן. אגענטיק סיסטעמען ארבעטן לויט צילן. ביישפיל: מען גיט אן אגענט דעם ציל צוצוגרייטן אן אנאליז פון קאנקורענטן; דער אגענט קען אליין זוכן דאטא, אנאליזירן טרענדס, ארויסברענגען תובנות, און שאפן א פארטיגן דאקומענט, מיט ווייניג אדער גארנישט מענטשלעכע איינמישונג. מען גייט איבער פון AI געטריבן פון באפעלן צו AI געטריבן פון צילן.

פלאנירן: די מעגלעכקייט צו פארשטיין וועלכע טריט מען דארף, אין וועלכן סדר, און מיט וועלכע כלים. דאס קען באדייטן צעטיילן א קאמפליצירטע אויפגאבע אין קלענערע, טראכטן טריט נאך טריט, און זיי דורכפירן

איינס נאכן אנדערן. אזוי קען מען לייזן פראבלעמען וואס מען קען נישט מיט איין פראמפט. דער משל: דער חילוק צווישן ענטפערן א פראגע און פארענדיגן א גאנצן פראיעקט.

זכרון: אן זכרון איז א AI-סיסטעם ווי א chatbot אן "שטאנד" (stateless), יעדער שמועס הייבט זיך אן פון נול. מיט זכרון פירן זיך אגענטן מער ווי געהילפן. סארטן זכרון:

- קורץ-טערמיניגער זכרון: האלט דעם קאנטעקסט פונעם איצטיגן שמועס און די טריט וואס גייען פאר.
- לאנג-טערמיניגער זכרון: היט אפ אינפארמאציע איבער באזונדערע סעסיעס, ווי וואס דער באניצער האט ליב אדער היסטארישע דאטא.
- וועקטאר-זכרון: באנוצט embeddings צו ארויסציען פאסיגע אינפארמאציע לויט ענלעכקייט, וואס מאכט מעגלעך פערזענליכקייט און באוואוסטזיין פונעם קאנטעקסט.

זכרון, ווערט געזאגט, איז דאס וואס מאכט אז אגענטן זאלן זיך פילן קלוג מיט דער צייט.

כלים, feedback-שלייפן און ארכיטעקטור (21:11:50)

באנוץ פון כלים איז איינע פון די שטארקסטע מעגלעכקייטן. אגענטן זענען נישט באגרענעצט צו זייער אינערליכן וויסן; זיי קענען ארבעטן מיט API'ס, דאטאבאזעס, זוך-מאשינען, ווייכווארג-אפליקאציעס און אויטאמאציע-פלאטפארמעס. ביישפיל: אנשטאט צו שאצן א חשבון, קען דער אגענט רופן א קאלקולאטאר-כלי און צוריקגעבן דעם פינקטלעכן רעזולטאט. דאס פארגרעסערט שטארק די פארלעסלעכקייט און די נוצן אין רעאלער צייט.

טראכטן און feedback-שלייפן: אגענטיק סיסטעמען ארבעטן אין איבערחזרנדיגע ציקלען: דער ציל, פלאנירן די קומענדיגע פעולה, זי דורכפירן, אָבסערווירן די רעזולטאטן, אפשאצן דעם אויסקום, און ענדערן די סטראטעגיע. דאס ערלויבט פאררעכטן טעותים, פארבעסערן און צופאסן באשלוסן. טראדיציאנעלע AI גיט איין רעזולטאט; אגענטיק AI פארבעסערט זיך אנגייענדיג צו איר ציל.

ארכיטעקטור: מערסטע אגענטיק סיסטעמען פאלגן א מאדולארער ארכיטעקטור, באשטאנדטיילן וואס ארבעטן צוזאמען:

- דער מוח: דער שפראך-מאדעל.
- דער זכרון: היט אפ קאנטעקסט און וויסן.
- די כלים: גיבן דרויסנדיגע מעגלעכקייטן.
- דער פלאנירער: באשליסט וואס צו טון ווייטער.
- דער דורכפירער: פירט אויס די פעולות.
- דער אפשאצער: קאנטראלירט די רעזולטאטן און גיט feedback.

דער צוגאנג מאכט די סיסטעמען פארברייטערבאר, גמיש און שטארק.

פארוואס דאס אלץ איז וויכטיג: די באשטאנדטיילן באשטימען ווי פארלעסלעך, זיכער און פארברייטערבאר א סיסטעם קען זיין. ווער עס טראכט נאר אין פראמפטס, וועט בויען chatbots; ווער עס טראכט אין אויטאנאמיע, פלאנירן, זכרון און כלים, וועט בויען אינטעליגענטע אגענטן. דאס, ווערט געזאגט, איז דער ענדערונג אין צוגאנג וואס פראפעסיאנאלן דארפן היינט. אין קומענדיגן קאפיטל גייט מען פון באגריפן צו אויספירונג.

קאפיטל דריי: פריימווערקס און ביבליאטעקן (21:14:00)

דער קורס הייבט אן דאס דריטע קאפיטל, וועגן פריימווערקס און ביבליאטעקן פאר אגענטן. נאך די יסודות פון פריערדיגן קאפיטל, אויטאנאמיע, פלאנירן, זכרון, כלים און טראכטן, קאנצענטרירט מען זיך דא אויף די פראקטישע סיסטעמען וואס ערלויבן פראגראמירער צו בויען אגענטן פאר דער רעאלער וועלט.

פריימווערקס, זכרון און פלאנירן ביי אגענטן (21:14:38)

LangChain: א מיטלשיכט צווישן מאדעלן און געצייג (21:14:38)

דער קורס ערקלערט אז אגענטישע אפליקאציעס ווערן געוועלך נישט געבויט פון גארנישט. פראגראמירער באניצן זיך מיט ספעציעלע פריימווערקס און ביבליאטעקן וואס גיבן פארטיגע טיילן פאר אָרקעסטראציע, צוזאמבינדן מיט געצייג, זכרון און אויספירונג.

דער ערשטער פריימווערק וואס ווערט באהאנדלט איז LangChain, איינער פון די מערסט באנוצטע פריימווערקס פאר אזעלכע אפליקאציעס. LangChain איז געמאכט כדי צו העלפן פראגראמירער פארבינדן גרויסע שפראך-מאדעלן (LLM) מיט דרויסנדיגע געצייג, API's, דאטא-באזעס, זכרון-סיסטעמען און דער לאגיק פון דער אפליקאציע.

דער הויפט-געדאנק: LangChain שטייט ווי א מיטלשיכט (middleware) צווישן די שפראך-מאדעלן, די געצייג און די אפליקאציעס. אנשטאט אליין שרייבן קאמפליצירטע אָרקעסטראציע-לאגיק, גיט LangChain מאָדולארע טיילן וואס מאכן די ארבעט גרינגער.

די הויפט-טיילן פון דער LangChain ארכיטעקטור:

- מאדעלן: די LLM's וואס מען באניצט פארן טראכטן און שאפן טעקסט, למשל מאדעלן פון OpenAI און Mistral AI. LangChain גיט "wrappers" כדי זיך צו פארבינדן מיט עטליכע פארשידענע פראוויידערס.
- פראמפטס: פראמפט-מוסטערן (prompt templates) גיבן דעם אריינגאנג א געארדנטע פארעם. דאס זיכערט אז די פראמפטס זענען אלעמאל גלייך, אז מען קען זיי נוצן נאכאמאל, און אז מען קען אריינלייגן דאטא דינאמיש.
- קייטן (Chains): א סדר פון שריט. למשל: מען הייבט אן פונעם אריינגאנג, דערנאך קומט די באארבעטונג, און צום סוף דער רעזולטאט. א קייט קען אנטהאלטן עטליכע פראמפטס, רופן פון געצייג און לאגישע שריט. קייטן פאסן פאר ארבעטס-פראצעסן וואס גייען אויף א פעסטן, פאראויס באשטימטן וועג.
- אגענטן: זיי ברענגען א מעגלעכקייט צו באשליסן, אנשטאט א פאראויס באשטימטן סדר. דער אגענט באשליסט וועלכע געצייג צו נוצן, ווען צו נוצן, און וואס צו טון נאכדעם. דא קומט ארויס די זעלבסטשטענדיגקייט.
- זכרון: זכרון-מאָדולן לאזן דעם אגענט געדענקען דעם קאנטעקסט פון איין שמועס צום צווייטן, למשל זכרון פונעם שמועס, זכרון פון א קיצור, זכרון אין א וועקטאר-דאטא-באזע, און נאך.
- געצייג: די זענען וואס מאכן פון אן AI אגענט א סיסטעם וואס קען טאקע טון זאכן.

אנדערע פריימווערקס אנשטאט LangChain (21:16:12)

כאטש LangChain איז פאפולער, זענען דא נאך פריימווערקס, יעדער מיט אייגענע מעלות:

• AutoGen, אנטוויקלט דורך Microsoft: לייגט דעם הויפט-טראָפ אויף צווזאמארבעט פון עטליכע אגענטן. זיינע מעלות: אגענטן קענען רעדן איינער מיטן אנדערן, סיסטעמען וואו יעדער אגענט האט א ראלע, ארבעטס-פראצעסן וואס גייען דורך א שמועס, און א פלאן וואס פאסט פאר גרויסע פירמעס. מען נוצט עס אסאך פאר פארשונג און פארגעשריטענער אויטאמאציע.

• CrewAI: געבויט ארום דעם געדאנק פון אגענטן מיט ראלעס וואס ארבעטן ווי א מאנשאפט. איר פיטשערס: איבערגעבן אויפגאבעס פון איין אגענט צום צווייטן, ארבעטס-פראצעסן מיט א היעראַרכיע, ספעציאליזירטע ראלעס ווי פארשער, שרייבער, אנאליטיקער, און גרינגע איינשטעלונג. CrewAI איז באזונדערס נוצלעך פאר אויטאמאציע פון געשעפט-פראצעסן.

• BabyAGI: איינער פון די פריערדיגע עקספערימענטאלע אגענט-פריימווערקס. ער האט אריינגעברענגט דעם געדאנק פון אויטאמאטיש שאפן אויפגאבעס, באשטימען וועלכע אויפגאבע איז מער וויכטיג, און א לופ פון אויספירונג וואס גייט ווייטער אן אויפהער. כאטש עקספערימענטאל, האט עס באאיינפלוסט אסאך היינטיגע פריימווערקס.

• AutoGPT: נאך א פיאנער-פראיעקט פון אויטאנאמישע אגענטן. זיינע מעגלעכקייטן: עס שרייבט זיך אליין פראמפטס, לופס וואס ווערן געטריבן פון א ציל, ארבעט אויפן אינטערנעט, און ארבעט מיט פיילס. AutoGPT האט געוויזן וויפיל מעגלעך איז מיט אינגאנצן זעלבסטשטענדיגע AI אגענט סיסטעמען.

געצייג פאר זכרון, פלאנירן און אויספירונג (21:17:42)

עס ווערט ערקלערט אז כדי צו בויען א גוטן אגענט דארף מען נאך טעכנאלאגיעס חוץ דעם פריימווערק.

זכרון-געצייג: זכרון ווערט אפט געמאכט מיט וועקטאר-דאטא-באזעס. פאפולערע ברירות: Chroma DB, Pinecone, Weaviate, FAISS. די סיסטעמען האלטן "embeddings" און לאזן ארויסנעמען לויט ענלעכקייט. זיי לאזן דעם אגענט געדענקען פריערדיגע שמועסן, ארויסנעמען רעלעוואנטן קאנטעקסט, און האלטן וויסן פאר א לאנגע צייט.

פלאנירונג-געצייג: זיי העלפן דעם אגענט צוטיילן א ציל אויף קלענערע אויפגאבעס. למשל: מאשינען וואס צוטיילן אויפגאבעס, פריימווערקס וואס פירן אויס לויט א גראף, און אָרקעסטראציע פון ארבעטס-פראצעסן. היינטיגע אגענט-פריימווערקס האבן אפט אריינגעבויטע פלאנירונג-מאָדולן אדער זיי פארבינדן זיך מיט אָרקעסטראציע-סיסטעמען.

אויספירונג-געצייג: אויספירן מיינט ארבעטן מיט דרויסנדיגע סביבות. געוועלכע פארבינדונגען: API's, Python פונקציעס, דאטא-באזעס, וועב-בראוזערס, און אויטאמאציע-פלאטפארמעס. די אויספירונג-שיכט איז קריטיש כדי צו מאכן פון AI א סיסטעם וואס ארבעט טאקע אין פראקטיק.

ווי אזוי ווערט דער LLM צווזאמגעפירט מיט געצייג (21:18:44)

אין הארץ פון אגענטישן AI ליגט אָרקעסטראציע: צווזאמשטימען דאס טראכטן פונעם LLM מיט דרויסנדיגע געצייג. א טיפישער פראצעס זעט אויס אזוי:

• דער באניצער גיט א ציל.

• דער LLM פארשטייט וואס מען וויל.

• דער אגענט באשליסט וועלכע געצייג צו נוצן און פירט אויס די אקציע.

• די רעזולטאטן קומען צוריק צום LLM.

- דער LLM שאצט אפ און פלאנירט דעם קומענדיגן שריט.

דער לופ גייט ווייטער ביז די אויפגאבע איז פארטיג. דאס שאפט א פארמאכטע לופ פון אינטעליגענץ.

פונקציע-רופן און געצייג-רופן: היינטיגע LLM's שטיצן א געארדנטע ארבעט מיט געצייג דורך פיטשערס ווי function calling, געצייג-סכעמעס (tool schemas) און API-פארבינדונגען. דאס לאזט אגענטן ארויסגיין פון בלויז שרייבן טעקסט צו טון אקציעס אין דער עכטער וועלט.

וועלכן פריימווערק צו קלייבן, און מגמות אין דער אינדוסטריע (21:19:46)

דער בעסטער פריימווערק הענגט אפ אין אייערע צילן. די סצענארן וואס וועלן געגעבן:

- פאר לערנען און עקספערימענטירן — LangChain.
- פאר עטליכע אגענטן צוזאמען — AutoGen.
- פאר געשעפט-ארבעטס-פראצעסן — CrewAI.
- פאר אויטאנאמישע עקספערימענטן — AutoGPT.
- פאר אויספארשן געדאנקען (concept exploration) — BabyAGI.

עס איז נישטא קיין איין בעסטע לייזונג; ארכיטעקטור-באשלוסן הענגען אפ אין די באדערפענישן.

מגמה אין דער אינדוסטריע — פון פראמפטס צו סיסטעמען: די אינדוסטריע גייט איבער פון "prompt engineering" צו "system engineering". פראגראמירער פלאנירן מער און מער AI סיסטעמען וואס באשטייען פון אסאך טיילן, אנשטאט בלויז איין שמועס מיט איין מאדעל. די מגמה שאפט א פארלאנג נאך פראפעסיאנאלן וואס פארשטייען AI אָרקעסטראציע, אגענט-ארכיטעקטור, צוזאמבינדן געצייג און פלאנירן אויטאמאציע.

פארוואס זכרון איז אזוי וויכטיג (21:20:48)

דער קומענדיגער קאפיטל באהאנדלט זכרון און לאנג-טערמיניגן קאנטעקסט, איינער פון די וויכטיגסטע טיילן פון אגענטישן AI. לויטן קורס איז זכרון דאס וואס שיידט טאקע אפ א פשוטע AI סיסטעם פון אן עכט אינטעליגענטן אגענט.

דער משל: אויב אן AI סיסטעם קען גארנישט געדענקען וועגן אייך, אייערע פריערדיגע שמועסן, אדער וואס עס האט געטון מיט 5 מינוט צוריק, פירט עס זיך ווי א "stateless" טשעטבאט, אן קיין מצב. אבער וועז מען לייגט צו זכרון, ווערט די סיסטעם פעאיג צו פערזענליכקייט, באוואוסטזיין פון קאנטעקסט, לערנען פון פריערדיגע אקציעס, און אפילו טראכטן אויף לאנגע צייט. מיט אנדערע ווערטער: זכרון פארוואנדלט AI פון רעאקטיווער אינטעליגענץ צו אינטעליגענץ וואס פאסט זיך צו.

אגענטישע סיסטעמען ארבעטן זעלטן אין איין שריט. זיי פירן אויס ארבעטס-פראצעסן פון עטליכע שריט, אמאל איבער עטליכע סעסיעס אדער אפילו לאנגע צייטן. כדי צו פונקציאנירן גוט, מוזן זיי געדענקען: פריערדיגע שמועסן, וואס דער באניצער האט ליב, די היסטאריע פון אויפגאבעס, צווישן-רעזולטאטן, און וויסן.

וועז אן AI אגענט האט זכרון, קען ער: האלטן א המשך אין שמועסן, נישט איבערחזרן טעותים, פארבעסערן באשלוסן מיט דער צייט, צופאסן ענטפערס פערזענליך, און אויספירן לאנגע אויפגאבעס עפעקטיוו. פון א סיסטעם-שטאנדפונקט, איז זכרון דאס וואס לאזט דעם אגענט וואקסן אנשטאט אנהייבן יעדעס מאל פון נול.

סארטן זכרון ביי אגענטן (21:22:19)

יעדער סארט זכרון האט אן אנדער ציל:

- באפער-זכרון (קורץ-טערמיניג): האלט די לעצטע שמועסן צייטווייליג און העלפט דעם אגענט האלטן דעם קאנטעקסט בשעת א שמועס אדער ארבעטס-פראצעס. חסרונות: ער איז געוועלך קליין צוליב דער באגרעניצונג פון "tokens", און וועז די סעסיע ענדיגט זיך ווערט ער אפט אויסגעמעקט. מען קען עס פארגלייכן מיטן מענטשליכן ארבעטס-זכרון — די אינפארמאציע וואס מען האלט פאר א קורצע צייט בשעת מען לייזט א פראבלעם. מען נוצט עס אין טשעט-שמועסן, אויפגאבעס פון טראכטן שריט ביי שריט, און פראצעסן וואס דארפן נאכפאלגן א צייטווייליגן מצב.
- קיצור-זכרון (summary memory): אנשטאט האלטן גאנצע שמועסן, דריקט די סיסטעם צוזאם די פריערדיגע שמועסן אין קורצע קיצורים — למשל וויכטיגע באשלוסן, וויכטיגע פאקטן אדער רעזולטאטן פון אקציעס, אנשטאט רויען טעקסט. מעלות: עס פארמינערט די קאמפיוטער-קאסטן, שפארט "tokens", און האלט פונדעסטוועגן דעם וויכטיגן לאנג-טערמיניגן קאנטעקסט. עס איז באזונדערס נוצלעך וועז אגענטן לויפן לאנגע צייטן און זאמלען אן אסאך אינפארמאציע.
- וועקטאר-דאטא-באזע זכרון (לאנג-טערמיניג): לויטן קורס דער וויכטיגסטער און שטארקסטער סארט. ער ארבעט אנדערש ווי געוועלכע דאטא-באזעס: אנשטאט האלטן פשוטן טעקסט פאר זוכן לויט שליסל-ווערטער, פארוואנדלט די סיסטעם דעם טעקסט אין נומעריש פארשטעלונגען וואס הייסן embeddings. די ווערן געהאלטן אין א וועקטאר-דאטא-באזע, און וועז דער אגענט דארף אינפארמאציע, נעמט ער ארויס רעלעוואנטע דאטא דורך זוכן לויט ענלעכקייט — לויטן באדייט, נישט לויט די גענויע ווערטער. דאס לאזט אגענטן געדענקען וואס באניצער האבן ליב, ארויסנעמען פארשונג-דאקומענטן, דערמאנען זיך אין אלטע שמועסן, און צוקומען צו וויסן וואס האט זיך אנגעזאמלט מיט דער צייט. מען קען עס פארשטעלן וי א לאנג-טערמיניגער אוצר פון וויסן פארן אגענט.

Chroma, Pinecone און דער פראצעס פון שטענדיגן זכרון (21:24:24)

כדי צו מאכן לאנג-טערמיניגן זכרון נוצן אגענטן וועקטאר-דאטא-באזעס. צוויי געוועלכע לייזונגען:

- Chroma: אסאך באנוצט פאר לאקאלער אנטוויקלונג און עקספערימענטן, ווייל עס איז open-source, לייכט, און גרינג צוצובינדן, באזונדערס מיט Python סביבות.
- Pinecone: א געפירטע לייזונג אין דער קלאוד, געמאכט פאר פראדוקציע-סיסטעמען. עס גיט מעגלעכקייט צו וואקסן, שנעלע ארויסנעמונג, און פארלעסלעכקייט אויף א פירמע-שטאפל.

דאס הייסט, לויט צי מען בויט א פראטאטיפ אדער א פראדוקציע-אפליקאציע, קען מען קלייבן אנדערע געצייג.

דער פראצעס פון שטענדיגן זכרון גייט אזוי:

- דאטא ווערט געשאפן אדער באקומען.
- דער טעקסט ווערט פארוואנדלט אין embeddings.
- די embeddings ווערן געהאלטן אין א וועקטאר-דאטא-באזע.
- וועז דער אגענט דארף קאנטעקסט, נעמט ער ארויס די מערסט רעלעוואנטע אינפארמאציע.

דאס לאזט אגענטן געדענקען אינפארמאציע אויף אן אומבאגרעניצטע צייט און נוצן וועז נויטיג.

Embeddings: דער יסוד פון וועקטאר-זכרון (21:25:25)

Embeddings פארוואנדלען טעקסט אין נומערישע וועקטארן וואס שטעלן פאר דעם באדייט. דאס מיינט אז זאצן מיט ענלעכן באדייט וועלן ארויסגעבן וועקטארן וואס זענען מאטעמאטיש נאענט איינער צום אנדערן. דער ביישפיל: "machine learning course" און "AI training program" וועלן מסתמא האבן ענלעכע embeddings, כאטש די וועדטער זענען אנדערש. Embedding-מאדעלן קען מען באקומען פון פראוויידערס ווי OpenAI און Hugging Face, און זיי טרייבן אסאך היינטיגע AI מעגלעכקייטן.

פארוואס זענען זיי אזוי שטארק? Embeddings לאזן מאכן זוכן לויטן באדייט, ארויסנעמען קאנטעקסט, פארשטארקן מיט וויסן, פערזענליכקייט, און רעקאמענדאציע-סיסטעמען. זיי זענען די יסוד-בלאקן פון היינטיגע אינטעליגענטע אפליקאציעס. אן embeddings וואלט לאנג-טערמיניגער זכרון אין AI סיסטעמען נישט געווען כמעט אזוי עפעקטיוו.

ווי זכרון פארבעסערט דעם אגענט, און די שווערigקייטן (21:26:26)

זכרון פארבעסערט דעם אגענט אין עטליכע ריכטונגען:

- באוואוסטזיין פון קאנטעקסט: דער אגענט קען זיך באציען צו פריערדיגע שמועסן און האלטן א המשך אן איבערטון ארבעט.
- פערזענליכקייט: ער פאסט זיך צו צו יעדן באניצער, געדענקט וואס דער האט ליב, און גיט צוגעפאסטע ענטפערס.
- עפעקטיווקייט און שנעלקייט: ווייל ער געדענקט פריערדיגע חשבונות, פארמיידט ער איבעריגע באארבעטונג, וואס פארמינערט קאסטן און ווארטן.
- בעסערע באשלוסן: ער קען לערנען פון פריערדיגע רעזולטאטן, פארמיידן טעותים און פארפיינערן סטראטעגיעס מיט דער צייט.
- לאנג-לויפנדיגע אויטאנאמישע סיסטעמען: געוויסע אגענטן ארבעטן שעהן אדער אפילו טעג, פאלגנדיג נאך ארבעטס-פראצעסן און פארשריט איבער סעסיעס. אן שטענדיגן זכרון קען עכטע זעלבסטשטענדיגקייט בכלל נישט עקזיסטירן.

אבער זכרון ברענגט אויך שווערigקייטן: מען דארף רעכענען מיט קאסטן פון האלטן דאטא; די גענויקייט פון ארויסנעמען מוז מען פירן פארזיכטיג; עס זענען דא פראגעס פון פריוואטקייט און זיכערהייט ווען מען האלט באניצער-דאטא; דער אגענט מוז גוט אויסזיפן דעם רעלעוואנטן קאנטעקסט; און פאלשער אדער פארעלטערטער זכרון קען אמאל פירן צו "hallucinations". דעריבער פארלאנגט פלאנירן א זכרון-ארכיטעקטור טיפע מחשבה.

פלאנירן און באשלוס-קייטן (21:28:29)

דער ווייטערדיגער קאפיטל באהאנדלט פלאנירן, נאך א הויפט-מעלה פון אגענטישן AI. פלאנירן איז וואס שיידט אפ א פשוטן AI געהילף פון אן עכט אינטעליגענטן אגענט. א פשוטער AI מאדעל וואלט אויף אנווייזונגען און גיט אן ענטפער, אבער אן AI אגענט מוז אליין אויסטראכטן ווי אזוי צו דערגרייכן א ציל שריט ביי שריט, אפט אן שטענדיגער מענטשליכער אנפירונג.

וואס איז פלאנירן? א פראצעס וואו דער אגענט פארשטייט דעם ציל, צעטיילט אים אויף קלענערע אויפגאבעס, באשטימט דעם סדר פון אקציעס, און פירט אויס די שריט בשעת ער פאלגט נאך דעם פארשריט — אנשטאט לייזן א פראבלעם אין איין ענטפער. דער טיפישער פלאנירונג-ציקל:

- דער אגענט באשטימט דעם ציל.
- ער אנאליזירט וואס מען דארף.
- ער שאפט א ליסטע פון אויפגאבעס.
- ער פירט זיי אויס.
- ער שאצט אפ די רעזולטאטן, און אויב נויטיג, טוישט ער דעם פלאן.

דער לופּ גייט ווייטער ביז דער ציל איז דערגרייכט. פלאנירן לאזט אגענטן באהאנדלען קאמפליצירטע ארבעטס-פראצעסן, פראבלעמען וואס פארלאנגען טראכטן אין עטליכע שריט, און לאנגע אויטאמאציע-אויפגאבעס וואס וואלטן געווען אוממעגלעך מיט איין פראמפט. צום סוף פון דעם אפשניט הייבט דער קורס אן צו רעדן וועגן צוויי וויכטיגע טראכט-סטראטעגיעס פאר פלאנירן: "chain of thought" קעגן "tree of thought".

פלאנירן, באנוץ-פעלער, אפשאצונג און די צוקונפט פון אגענטן (21:30:01)

Chain of thought קעגן tree of thought (21:30:01)

דער קורס הייבט אן מיט צוויי באגריפן וואס מען הערט אפט ביים רעדן וועגן ווי אן AI אגענט טראכט: chain of thought (אויך קייט פון געדאנקען) און tree of thought (בוים פון געדאנקען).

Chain of thought אַרבעט דורך א גלייכע ליניע פון טריט. מען קען זיך עס פארשטעלן ווי מען גייט פון טריט איינס צו טריט צוויי, פון צוויי צו דריי, ביז מען קומט אן צום לעצטן ענטפער. דער צוגאנג איז פארהעלטניסמעסיג פשוט, אבער עפעקטיוו. ער פאסט גוט וועו דער וועג צום ענטפער איז פאראויס באקאנט, למשל ביי מאטעמאטישע חשבונות אדער ביי סטרוקטורירטע לאגישע פראבלעמען.

אבער עס זענען דא חסרונות. ווייל די טראכטונג גייט נאר אויף איין וועג, קען א טעות אין איין טריט זיך ווייטער פארשפרייטן איבער דער גאנצער קייט, און מען זוכט נישט קיין אנדערע מעגליכע לייזונגען. דערפאר, הגם chain of thought איז שטארק, איז עס נישט אלעמאל גענוג בייגזאם פאר קאמפליצירטע באשלוסן.

Tree of thought בריייטערט אויס דעם פראצעס אויף עטליכע מעגליכע וועגן. אנשטאט זיך צו באשליסן גלייך אויף איין ריכטונג, פרובירט דער אגענט אויס פארשידענע אפציעס, שאצט זיי אפ, און קלייבט דערנאך אויס די בעסטע. דאס פארבעסערט די קוואליטעט פון באשלוס, ווייל מען קען פארגלייכן איין אלטערנאטיוו מיט דער אנדערער. עס איז באזונדערס נוצלעך ביי קאמפליצירטע פלאנירונגען וואו עס קענען זיין עטליכע סטראטעגיעס. לויט דעם קורס איז א בוים-ארטיגע טראכטונג נענטער צו ווי א מענטש טראכט וועו ער לייזט סטראטעגישע פראבלעמען.

דער איינפאכסטער וועג צו פארשטיין דעם חילוק:

- Chain of thought – גלייכע, ליניארע טראכטונג.
- Tree of thought – פארצווייגטע טראכטונג.

אין עכטע אגענט-סיסטעמען ווערן ביידע צוגאנגען אפט קאמבינירט, לויט ווי קאמפליצירט די ארבעט איז.

צעטיילן די ארבעט און פלאנירן דעם ארבעטס-פלוס (21:31:31)

כדי אן AI אגענט זאל קענען דערגרייכן גרויסע צילן, מוז ער צעברעכן א גרויסן ציל אויף קלענערע טיילן וואס מען קען טאקע אויספירן. דער פראצעס הייסט task decomposition) צעטיילן די ארבעט).

היינטיגע אגענט-פריימווערקס לאזן פראגראמירער פלאנירן ארבעטס-פלוסן וואס זענען נישט נאר ליניאר, נאר דינאמישע סיסטעמען וואס פאסן זיך צו לויט די באדינגונגען. א שטארקער באגריף דא איז graph-based workflows: יעדע ארבעט ווערט אפגעטיילט און דארגעשטעלט ווי א "נאוד" (node), און די נאודס זענען פארבונדן דורך וועגן פון באשלוסן. דאס מאכט מעגליך:

- אויספירן א טריט נאר אויב א געוויסער תנאי איז מקוים;
- רינגען (loops) וואס חזרן איבער א טריט;
- אפשטעל-פונקטן (checkpoints) וואו א מענטש דארף געבן א הסכמה;
- און אפילו אויספירן עטליכע ארבעטן גלייכצייטיג.

אין פארגלייך מיט פעסטע, פאראויס-באשטימטע "פייפליינס", גיט א גראף-באזירטע פלאנירונג אסאך מער בייגזאמקייט און פארלעסלעכקייט.

מערסטעפיגע זעלבסטשטענדיגע ארבעט (21:32:01)

דער קורס ערקלערט וואס "multi-step autonomy" מיינט אין דער פראקטיק. א טיפישער זעלבסטשטענדיגער ארבעטס-פלוס זעט אויס אזוי:

- דער אגענט באקומט א ציל;
- ער שאפט א פלאן;
- ער קלייבט אויס דעם פאסיגן געצייג;
- ער פירט אויס די אקציעס;
- ער שאצט אפ די רעזולטאטן;
- אויב נויטיג, דערהיינטיגט ער דעם פלאן.

דער ציקל חזרט זיך איבער ביז דער ציל איז דערגרייכט.

א ביישפיל: דער ציל איז צו מאכן א באריכט פון מארק-פארשונג. דער אגענט קען אליין זוכן דאטא וועגן דער אינדוסטריע, ארויסנעמען וויכטיגע אינפארמאציע, אנאליזירן די קאנקורענטן, שאפן דעם באריכט און עס שיין צוגרייטן – אלעס מיט ווייניג מענטשלעכער באטייליגונג. דאס איז אסאך מער פארגעשריטן ווי פשוט ענטפערן אויף א פראמפט; דאס איז עכטע זעלבסטשטענדיגע אויספירונג.

שוועריגקייטן ביים פלאנירן (21:33:05)

פלאנירן איז נישט גרינג, און עס זענען דא עטליכע טעכנישע שוועריגקייטן וואס פראגראמירער מוזן נעמען אין חשבון. אן אגענט קען צעטיילן די ארבעט אויף א פאלשן אופן, אויסקלייבן נישט די ריכטיגע געצייג, אדער פארכאפט ווערן אין א רינג אן א סוף. עס קענען אויך זיין באגרעניצונגען פון רעסורסן, אדער אומזיכערקייט ביים באשליסן.

כדי צו בויען פארלעסלעכע סיסטעמען, לייגן פראגראמירער אריין שוץ-מיטלען, ווי:

- א גרענץ אויף צאל טריט;
- קאנטראל-בדיקות פון רעזולטאטן;
- מיטלען צו נאכקוקן דעם פראצעס;
- שטאפלען וואו א מענטש דארף געבן א הסכמה.

פלאנירן איז שטארק, אבער עס פארלאנגט א פארזיכטיגן פלאן כדי עס זאל ארבעטן זיכער אין א פראדוקציע-סביבה.

פלאנירן – א יסוד-פעאיגקייט פונעם אגענט (21:33:36)

צום סוף פון דעם טייל ווערט באטאנט אז פלאנירן פארוואנדלט AI סיסטעמען פון בלויז "רעאגירן" צו סטראטעגישע פראבלעם-לייזער. עס ערמעגליכט אגענטן צו לייזן קאמפליצירטע פראבלעמען, אויספירן לאנגע ארבעטס-פלוסן, זיך צופאסן צו נייע אינפארמאציע, און ארבעטן מיט א הויכן שטאפל זעלבסטשטענדיגקייט. דער קורס פאסט עס צוזאם: אויב זכרון (memory) גיט דעם אגענט ערפארונג, גיט פלאנירן אים ריכטונג. צוזאמען שאפן די צוויי פעאיגקייטן דעם יסוד פון אינטעליגענטער AI אויטאמאציע.

פארגעשריטענע באנוץ-פעלער: פארשונג-געהילפן (21:34:07)

א נייער קאפיטל הייבט אן: פארגעשריטענע באנוץ-פעלער פון agentic AI. ביז יעצט האט מען גערעדט וועגן די בוי-שטיינער – זכרון, פלאנירן, טראכטן, און פארבינדונג מיט געצייג. יעצט קומט די גרויסע שאלה: וואו ווערט דאס אלעס טאקע גענוצט אין דער עכטער וועלט? אין דעם קאפיטל קוקט מען אויף פעלער וואו אגענט-סיסטעמען קאמבינירן די פעאיגקייטן כדי אויסצופירן קאמפליצירטע ארבעטס-פלוסן מיט ווייניג מענטשלעכער באטייליגונג.

איינע פון די שטארקסטע באנוצונגען איז זעלבסטשטענדיגע פארשונג-געהילפן. אנדערש ווי טראדיציאנעלע געצייג וואס ברענגען פשוט אינפארמאציע, קענען אגענט-סיסטעמען זוכן אויפן אינטערנעט, ארויסנעמען דעם רעלעוואנטן אינהאלט, פארגלייכן עטליכע מקורות, און שאפן סטרוקטורירטע צוזאמפאסונגען, אפט מיט אנווייזונגען צו די מקורות. דאס איז זייער נוצלעך פאר מארק-פארשונג, אקאדעמישע פארשונג, אנאליז פון קאנקורענץ, און אפילו שאפן אינהאלט. דער הויפט מעלה איז נישט נאר דאס געפינען אינפארמאציע, נאר דאס אנהאלטנדיגע טראכטן און באשטעטיגן איבער עטליכע טריט: דער אגענט געפינט נישט בלויז דאטא, ער שאצט זי אפ.

עטליכע אגענטן ארבעטן צוזאמען (21:35:40)

א צווייטער פארגעשריטענער באגריף איז multi-agent collaboration – מיטארבעט פון עטליכע אגענטן. אנשטאט זיך צו פארלאזן אויף איין אגענט וואס זאל טון אלעס, קען מען בויען סיסטעמען וואו עטליכע ספעציאליזירטע אגענטן ארבעטן צוזאמען, ענלעך צו א מענטשלעכער מאנשאפט. פריימווערקס ווי CrewAI ערמעגליכן אזא ראלע-באזירטע ארכיטעקטור. למשל:

- א פארשונג-אגענט וואס זאמלט דאטא;
- אן אנאליטיקער-אגענט וואס ארבעט אויס די אינפארמאציע;
- א שרייבער-אגענט וואס שאפט דעם אינהאלט;

- א רעצענזענט-אגענט וואס באשטעטיגט דאס לעצטע רעזולטאט.

דער צוגאנג פארבעסערט סיי די קוואליטעט סיי די מעגליכקייט צו וואקסן, ווייל יעדער אגענט קאנצענטרירט זיך אויף איין באשטימטע אחריות – פונקט ווי עכטע ארגאניזאציעס ארבעטן.

אויטאמאציע-פייפליינס און פארבינדונג מיט דרויסנדיגע געצייג (21:36:11)

Agentic AI ערמעגליכט אויך אויטאמאציע פון אנהייב ביזן סוף, למשל ביי אימעילס, פארשונג און צוזאמפאסונג. מען זאל זיך פארשטעלן א סיסטעם וואס לייענט אריינקומענדיגע מעסעדזשעס, נעמט ארויס די וויכטיגע אינפארמאציע, מאכט א הינטערגרונט-פארשונג, שאפט אן ענטפער און שיקט אים אוועק אויטאמאטיש. דער גאנצער פלוס קען לויפן מיט ווייניג מענטשלעכער השגחה. אזעלכע פייפליינס פארמינערן שטארק די האנט-ארבעט און פארגרעסערן די פראדוקטיוויטעט אין ביזנעס, באזונדערס ביי ארבעטן וואס זענען באזירט אויף וויסן.

AI אגענטן ווערן טאקע שטארק ווען זיי פארבינדן זיך מיט עכטע ווייכווארג-סיסטעמען:

- Zapier – לאזט אגענטן אנצינדן ארבעטס-פלוסן איבער טויזנטער אפליקאציעס;
- Notion – פאר פארוואלטן וויסן און דערהיינטיגן דאקומענטאציע;
- Google Sheets – פאר אפהיטן דאטא, שאפן באריכטן און האלטן סטרוקטורירטע רעקארדס.

דורך די פארבינדונגען קענען אגענטן דערהיינטיגן דאטא-באזעס, סינכראניזירן אינפארמאציע צווישן פלאטפארמעס, שאפן אנאליטיק און אויטאמאטיזירן אפעראציע-פראצעסן. דא ווערט AI פון א שמועס-געהילף א טיפע שיכט פון אפעראציאנעלער אויטאמאציע אין ארגאניזאציעס.

מעסטן ווי גוט דער אגענט ארבעט (21:37:42)

דער נעקסטער קאפיטל באהאנדלט אפשאצונג פון אגענטן און פארבעסערן פראמפטס. בויען אן אגענט איז נאר דער אנהייב; די עכטע אויפגאבע איז אים מאכן פארלעסלעך, עפעקטיוו און גרייט פאר פראדוקציע. אן אגענט וואס ארבעט איינמאל איז נישט גענוג – מען דארף אגענטן וואס ארבעטן קאנסעקווענט.

דער ערשטער טריט צו פארבעסערן אן אגענט איז פארשטיין ווי גוט ער ארבעט. די הויפט-שאלה: האט די סיסטעם דערגרייכט איר ציל, און ווי עפעקטיוו? אפט גענוצטע מאסן:

- וויפל מאל דער אגענט ענדיגט ערפאלגרייך די ארבעט (task completion rate);
- די גענויקייט פון די רעזולטאטן;
- די צאל טריט וואס ער האט גענומען ביז ער איז אנגעקומען צו א לייזונג;
- צי די געציג זענען גענוצט געווארן ריכטיג;
- וויפל צייט אדער געלט די אויספירונג קאסט.

אין דער עכטער וועלט פרובירן פראגראמירער אגענטן אויס מיט פאראויס-צוגעגרייטע סצענארן אדער בענטשמארק-אויפגאבעס וואס סימולירן פראדוקציע-באדינגונגען, כדי צו געפינען שוואכקייטן פארן ארויסלאזן.

פראמפט-דיבאגינג און פידבעק (21:39:15)

א זייער פראקטישע קונסט ביי אגענטן איז "דיבאגן" פראמפטס. פראמפטס זענען די אנווייזונגען וואס פירן דעם אגענט'ס טראכטן; אויב זיי זענען נישט קלאר אדער שלעכט געבויט, וועט דער אגענט זיך מוטשען. עפעקטיווע מיטלען:

- צעברעכן גרויסע פראמפטס אויף קלענערע, מער באשטימטע אנווייזונגען;
- צולייגן באגרענעצונגען און פארמאט-פאדערונגען, כדי צו פארבעסערן די קאנסעקווענץ;
- געבן ביישפילן (few-shot prompting), וואס העלפט דעם מאדעל קלארער פארשטיין וואס מען ערווארט;
- קלאר דעפינירן דעם אגענט'ס ראלע און ציל;
- פארשרייבן (logging) די צווישן-טריט פון טראכטן, כדי צו זען ווו עס פאלט דורך.

דער קורס פארגלייכט עס מיט דיבאגן קאוד: קליינע ענדערונגען קענען ברענגען איבערראשנד גרויסע פארבעסערונגען.

אגענטן קענען זיך אויך פארבעסערן דורך פידבעק, וואס קומט פון פארשידענע מקורות: באניצער פאררעכטן רעזולטאטן, סיסטעמען פארשרייבן צי ארבעטן זענען געלונגען אדער נישט, אפשאצונגען, ראנגירונגען, און לאגס וואס כאפן ארויס דאטא ווען דער ארבעט. ווען מען ברענגט אריין דעם פידבעק אין דער סיסטעם, קענען אגענטן ענדערן סטראטעגיעס, פארפיינערן פראמפטס און פארמינערן טעותים מיט דער צייט – אזוי שאפט מען זיך-צופאסנדיגע סיסטעמען וואס פארבעסערן זיך אן אויפהער אנשטאט צו בלייבן שטיין.

Prompt engineering קעגן fine-tuning (21:40:47)

ביים אפטימיזירן אגענטן זענען דא צוויי הויפט-צוגאנגען. Prompt engineering מיינט ענדערן די אנווייזונגען צו ענדערן דעם מאדעל אליין; עס איז שנעלער, ביליגער, און ווירקזאם ביי רוב פעלער, און דערפאר פארלאזן זיך רוב אגענט-סיסטעמען שטארק אויף פראמפט-פלאנירונג. Fine-tuning, אויף דער אנדערער זייט, מיינט טריינירן דעם מאדעל אויף אייגענע דאטא-סעטס; עס קען געבן א מער קאנסעקווענט פירונג אין ספעציאליזירטע געביטן, אבער עס קאסט מער און איז קאמפליצירטער. אין דער פראקטיק פרובירן פראגראמירער אויס אלע מעגליכקייטן פון prompt engineering פאר זיי גייען איבער צו fine-tuning.

זעלבסטשטענדיגקייט קעגן קאנטראל (21:41:47)

דער לעצטער קאפיטל פונעם קורס באהאנדלט די סכנות, עטישע פראגן און די צוקונפט פון agentic AI; אויך וועט מען בויען אן אגענט. וועו די סיסטעמען ווערן מער זעלבסטשטענדיג און שטארקער, שטעלט זיך די שאלה ווי צו נוצן די טעכנאלאגיע מיט אחריות – ווייל מיט גרעסערער פעאיגקייט קומט גרעסערע אחריות.

איינע פון די גרעסטע מעלות פון agentic AI איז זעלבסטשטענדיגקייט, אבער די זעלבע זעלבסטשטענדיגקייט ברענגט סכנות:

- אן אגענט קען מאכן פאלשע באשלוסן לויט אומפולשטענדיגע אינפארמאציע;
- ער קען אויספירן אקציעס ווען ער האט נישט ריכטיג פארשטאנען די אנווייזונגען;
- ארגאניזאציעס קענען ווערן צו אפהענגיג פון אויטאמאציע;
- עס קענען זיין זיכערהייט-לעכער ווען אגענטן ארבעטן מיט דרויסנדיגע סיסטעמען אן פאסיגע שוץ-מיטלען.

דערפאר פאדערט א זיכערע ארויסלאזונג שטארקע השגחה, קאנטראל-מיטלען און מענטשלעכע אויפזיכט. זעלבסטשטענדיגקייט טאר קיינמאל נישט מיינען פעלן פון קאנטראל; דער ציל איז "קאנטראלירטע זעלבסטשטענדיגקייט".

Alignment און hallucination (21:42:49)

צוויי גרויסע טעכנישע שווערigkייטן זענען alignment און hallucination. Alignment מיינט זיכער מאכן אז דעם אגענט'ס אקציעס שטימען מיט מענטשלעכע כוונות און צילן; אפילו א זייער פעאיגע סיסטעם קען שאפן פראבלעמען אויב זי שטרעבט צום פאלשן ציל. Hallucination איז וועו AI שאפט פאלשע אדער אויסגעטראכטע אינפארמאציע, און קלינגט דערביי זעלבסטזיכער – דאס איז באזונדערס געפערלעך אין ערנסטע געביטן ווי געזונט, פינאנצן אדער געזעץ. כדי צו פארמינערן די סכנות, לייגן פראגראמירער אריין באשטעטיגונג-טריט, באשטעטיגונג דורך געצייג, פידבעק-רינגען, און אפשטעל-פונקטן מיט א מענטש אין מיטן (human-in-the-loop). פארלעסלעכקייט איז נישט קיין לוקסוס – זי איז נייטיג פאר עכטן באנוץ.

די צוקונפט און לעצטע געדאנקען (21:43:49)

לויט דעם קורס ווערט ערווארט אז agentic AI וועט איבערמאכן ווי מען לעבט און ארבעט. די הויפט-טרענדס:

- פערזענלעכע AI געהילפן וואס פירן א לוח, ארבעטן, און העלפן ביי באשלוסן;
- אויטאמאציע אין דער ארבעט, וואו אגענטן טוען איבערחזורנדיגע ביזנעס-אפעראציעס, און מענטשן קענען זיך קאנצענטרירן אויף ווערטפולערע ארבעט;
- זעלבסטשטענדיגע פארשונג-סיסטעמען וואס שנעלערן אויף וויסנשאפטלעכע אנטדעקונגען דורך אנאליזירן דאטא און פארשלאגן השערות;
- און, אפשר די וויכטיגסטע, סיסטעמען וואו מענטשן און אגענטן ארבעטן צוזאמען ווי איינע מאנשאפט.

מען גייט אריבער פון AI ווי א געצייג צו AI ווי א דיגיטאלע ארבעטס-קראפט.

צום סוף: מען האט דורכגעגאנגען א פולן וועג – פון די יסודות, דורך ארכיטעקטור, פלאנירן, זכרון, ביז פארגעשריטענע באנוצונגען. אבער דאס לערנען הערט נישט אויף. דער בעסטער וועג צו באהערשן דאס געביט איז דורך פראקטישע עקספערימענטן: בויען אייגענע אגענטן, זיי פארבינדן מיט געצייג און APIs, אויספרובירן סיסטעמען פון עטליכע אגענטן וואס ארבעטן צוזאמען, און עקספערימענטירן מיט טעכניקן פון אפשאצונג און אפטימיזאציע. פראקטישע ערפארונג איז דאס וואס פארוואנדלט וויסן אין מומחיות.

בויען אן AI אגענט פאר אפוינטמענטס מיט N8N (21:45:20)

אריבערגאנג: פון פאסיווע מאדעלן צו אגענטן (21:45:20)

דער קורס ענדיגט דעם פריערדיגן טייל מיט א וויכטיגן געדאנק: מען רוקט זיך אוועק פון פאסיווע מאדעלן וואס גיבן בלויז ענטפערס, צו אויטאנאמע מאדעלן וואס קענען טראכטן, פלאנירן און טון אקציעס. עס ווערט באטאנט אז די צוקונפט וועט געהערן צו די פראפעסיאנאלן וואס פארשטייען נישט נאר פראמפטס, נאר די גאנצע פלאנירונג פון אן AI סיסטעם.

דערנאך הייבט מען אן בויען אן AI אגענט. דער ציל: אן אויטאמאטישער אימעיל-שרייבער, אן AI אגענט וואס שיקט באשטעטיגונגען צו מענטשן וואס בעטן אן אפוינטמענט. עס ווערט ערקלערט אז דאס איז גאנץ פשוט.

די פלאטפארמעס וואס מען דארף (21:45:50)

מען וועט ניצן צוויי הויפט-זאכן: N8N און Google. פון Google דארף מען:

- Google Cloud
- Google Sheets
- Google Forms
- Gmail

דער פלאן איז צו נעמען API's פון די פלאטפארמעס און זיי צונויפבינדן. דערנאך שרייבט מען עטליכע פראמפטס, כדי דער AI אגענט זאל פארשטיין וואס די מענטשן וואס בעטן אן אפוינטמענט פאדערן, און נאכדעם גייט מען ווייטער מיטן אויפבויען.

ערשטנס דארף מען עפענען א חשבון אויף דער N8N פלאטפארמע און אים באשטעטיגן. אין דער לעקציע איז די באשטעטיגונג שוין געמאכט. דערנאך לאגט מען זיך אריין אין Google Cloud, מען עפנט א חשבון און מען גייט אריין אין דער Console. מען איז אויך אריינגעלאגט אין Google Sheets , און מען וועט אויפמאכן א נייע Google Form.

אויפשטעלן די Google Form (21:46:52)

מען הייבט אן מיט א ליידיגער פארם (Blank Form). נאכן שאפן קען מען איר געבן א נאמען, למשל "My Appointments". די פארם וועט זיין אפן פאר יעדן וואס זוכט אן אפוינטמענט.

דערנאך לייגט מען צו פעלדער:

- נאמען: א Short Answer, און עס מוז זיין Required. וועז דער באניצער פילט נישט אויס זיין נאמען און די פרטים, קען מען ווייטער גארנישט טון.
- אימעיל: אויך Required. דאס איז דער וויכטיגסטער פעלד, ווייל מען וועט שיקן אימעילס, און לויט דעם וואס דער באניצער שרייבט דא שיקט מען דעם אימעיל.
- טעלעפאן-נומער: אויך א Short Answer און Required. עס ווערט אבער באמערקט אז מען קען דעם פעלד איבערלאזן, ווייל מען שיקט דא נישט קיין מעסעדזשעס. וועז מען וואלט געניצט טריגערס ווי Telegram, וואלט מען עס געקענט ניצן.
- דאטום: מען וועלט דעם Date פארמאט.

עס ווערט געווארנט אז מען זאל זיך פארזיכערן אז דער דאטא-טיפ פון יעדן פעלד איז ריכטיג.

אין די איינשטעלונגען נעמט מען נאר באשטעטיגטע (Verified) רעספאנסעס, און מען קען אויך שיקן א קאפיע צו די וואס פילן אויס די פארם, וועז זיי בעטן דאס.

פארבינדן די פארם מיט א Google Sheet (21:48:25)

מען קען די פארם פארבינדן מיט א Google Sheet. מען גייט אריין אין Responses און מען קליקט אויף "Link to Sheet". מען גיט דעם שיט א נאמען, דא ווידער "My Appointments", דער זעלבער נאמען ווי די פארם. מען זעט אז אלע פעלדער וואס מען האט אריינגעלייגט אין דער פארם באקומען די פארם פון קאלאמס אין Google Sheet.

פארן ארויסגעבן (Publish) די פארם קען מען זיך נאכקוקן די פרטים, און מען קען אויך שפעטער צוריקקומען און זי רעדאגירן וועז מען וויל.

דער טריגער אין N8N (21:48:57)

צוריק אויף N8N, קליקט מען אויף דעם ערשטן שריט, דאס זענען די טריגערס. עס איז דא א ברייטע אויסוואל: א Chat, דורכלויפן אן Evaluation, לויט א צייט-פלאן (On Schedule), אן App Event, אדער מען טריגערט מיט דער האנט (Manually).

דא וויל מען Google Sheets אלס טריגער. עס זענען דא אפציעס ווי On Row Added, On Row Updated , און On Row Added or Updated. מען קען אויך זען נאך אקציעס: Create Sheet, Delete Sheet, Append Row, Update Row, Append or Update Row א.א.וו. מען וועלט לויט דעם וואס מען דארף. דא איז On Row Added גענוג: וועז איינער פילט אויס די פארם און א נייע רייע ווערט אנגעפילט אינעם שיט, זאל די ריי ווערן געזומען אין באטראכט און טריגערן דעם וואָרקפלאו.

אבער מען דארף צולייגן א Credential , און מען האט נאך קיינע נישט. דא דארף מען זיך אריינלאגן אין Gmail א.א.וו. מען זעט אויך א Mock Output, וואס מען קען זיך באקוקן פארן צולייגן די Credentials.

כדי צולייגן א Credential דארף מען OAuth אויטענטיפיקאציע, אדער מען לאגט זיך איריין דורך Google. פריער אבער מוז מען דורכפירן די OAuth פאדערונגען אין Google Cloud.

שאפן א פראיעקט אין Google Cloud (21:50:30)

מען גייט אריין אין דער Google Cloud Console. דארט קען מען וועלן א פראיעקט אדער שאפן א נייעם. ווייל מען האט נאך קיין פראיעקט נישט, שאפט מען איינעם פון אנפאנג אן. ערשט מוז מען מסכים זיין צו געוויסע תנאים, אנגעבן דאס לאנד און נאך פרטים, און קליקן "Agree and Continue".

מען וועט שפעטער ניצן APIs & Services, אבער פריער דארף מען א נייעם פראיעקט. מען קליקט "Select a Project". ווייל מען מאכט עס פאר לערן-צווועקן, וועלט מען "No Organization". מען גיט דעם פראיעקט א נאמען, למשל "My Appointments" אדער וועלכן נאמען מען וויל; אין דער לעקציע הייסט ער "My First Project". מען קליקט Create, און ער ווערט געשאפן אין הינטערגרונט.

נאכדעם וועלט מען דעם פראיעקט און מען זעט דעם דעשבאָרד: API, Cloud Platform, Project Info , די Resources וואס מען ניצט, Monitoring א.א.וו.

דער OAuth Consent Screen און שאפן א Client (21:52:03)

פון APIs & Services גייט מען צום OAuth Consent Screen. דארט זעט מען וואס מען דארף שאפן: צוויי זאכן, א Client און אן Audience.

ערשט שאפט מען א Client. מען סקראלט אביסל אראפ ביזן אונטערשטן טייל און מען קליקט "Create Client", און דערנאך "Get Started". מען גיט אינפארמאציע וועגן דער אפליקאציע. מען קען ניצן דעם זעלבן נאמען ווי דער שיט אדער די פארם; דא הייסט זי "Appointment App". מען פארבינדט אן אימעיל-אדרעס.

מען פרעגט צי דאס איז פאר אן אינערלעכן (Internal) אדער אויסערלעכן (External) עולם. מען וועלט External , ווייל יעדער וואס זוכט אן אפוינטמענט מוז עס קענען ניצן, און מען קען אים שיקן דעם לינק. ביי

Contact Information שרייבט מען דעם אימעיל מיט וועלכן מען האט געשאפן דעם פראיעקט. דערנאך Finish, Continue און Create. דער Audience איז געשאפן.

איינשטעלן דעם OAuth Client און די URI (21:53:03)

דערנאך קליקט מען אויף OAuth Client. מען וועלט דעם טיפ "Web Application". מען קען שאפן פאר Android, א Chrome Extension א.א.וו., אבער מען וועלט Web Application, כדי עס זאל פאסן סיי אויף מאבייל סיי אויפן דעסקטאפ.

מען דארף צולייגן א URI. די URI איז גארנישט אנדערש ווי דער URL פון דער N8N פלאטפארמע. מען קאפירט דעם URL פון HTTP, און מען לאזט אוועק אלעס וואס קומט נאך "workflow": דאס שטריכל און אלע קאודס אדער IDs פונעם פראיעקט. די לעקציע פארריכט זיך שפעטער אז מען דארף קאפירן נאר ביז "cloud". דעם טריגער אין N8N קען מען פארמאכן אדער שפייכלערן אזוי ווי ער איז, אן רעדאגירן. מען קאפירט דעם URL, גייט צוריק צו Google Cloud און מען לייגט אים אריין. עס וועלט באמערקט אז ביי דער ערשטער URI דארף מען נישט קיין שטריכל אינעם סוף.

דערנאך דארף מען אויספילן Authorized Redirect URIs. דא שרייבט מען נאכן שטריכל נאך א טייל:

rest/oauth2-credential/callback

עס וועלט געזאגט אז דער ארבעטס-בוך פון דער לעקציע איז פארלינקט אין דער Description, כדי מען זאל עס קענען קאפירן וועט מען וויל איבערמאכן דעם פראיעקט.

Client ID און פארבינדן מיט N8N (21:55:05)

נאכן שאפן דעם Client באקומט מען א Client ID. מען קען אים קאפירן ערגעץ פאר שפעטערדיגע באנוץ, אדער אראפלאדן אלס JSON. אין דער לעקציע טוט מען ביידע: מען לייגט דעם Client ID אריין אין א Notepad דאקומענט אונטערן נאמען "Client ID", און מען לאדט אראפ דעם JSON טעקע.

דערנאך גייט מען צוריק צו N8N און לייגט צו א נייע Credential. מען וועלט OAuth, וואס וועלט רעקאמענדירט. מען לאגט זיך אריין און וועלט דעם Google חשבון מיט וועלכן מען האט געשאפן דעם פראיעקט און אלעס וואס מען האט געמאכט אין Google Cloud Console. אזוי זענען ביידע, Google Cloud און N8N, פארבונדן צו דער זעלבער ID, און די פארבינדונג איז געלונגען. דער טריגער איז כמעט גרייט.

צולייגן טעסט-באניצער אינעם Audience (21:56:06)

מען דארף נאך עטליכע זאכן, ווי דעם שיט און זיינע קאלאמס. אבער ביז איצט האט מען נאר געשאפן דעם OAuth Client; מען דארף נאך אויפשטעלן דעם Audience. מען קליקט אויף Audience און מען מוז ארויסגעבן (Publish) די אפליקאציע, פונקט ווי מען האט געטון ביים Client.

פריער דארף מען צולייגן Test Users. מען קען געבן וועלכע אימעיל-אדרעס מען וויל. אין דער לעקציע לייגט מען צו אן אנדערן אימעיל אלס טעסט-באניצער, און צום זיכערקייט אויך דעם Gmail מיט וועלכן מען האט געשאפן דעם פראיעקט, אזוי אז מען האט ביידע. עס קומט ארויס א מעלדונג אז געוויסע חשבונות זענען נישט בארעכטיגט; עס וועלט ערקלערט אז מען קען עס איגנארירן. מען קליקט Save. עס קומט נאך א וואַרנונג, וואס מען קען אויך איגנארירן, ווייל מען אַרבעט נאר אין Testing Status.

צוריק צו N8N: וועלן דעם שיט (21:57:06)

צוריק אויף N8N, קען מען איינשטעלן די Mock Data, דאס הייסט ווי אזוי דער אינפוט וועט אנקומען. מען זעט אז די פעלדער פון דער פארם (נאמען, אדרעס א.א.וו.) זענען דא נאך נישטא. דעריבער דארף מען וועלן דעם דאקומענט, דעם "My Appointments" שיט. דאס איז דער שיט וואס מען וועט ניצן ווייטער.

אויטאמאטישע באשטעטיגונג פאר אפוינטמענטס מיט n8n (21:57:39)

פארבינדן דעם טריגער מיט דעם ריכטיגן בלאט (21:57:39)

דער קורס האלט ביי בויען אן אויטאמאציע-ארבעטספלוס אויף דער פלאטפארמע n8n. דער ציל: ווען א מענטש פילט אויס א Google Form פאר אן אפוינטמענט, זאל ער באקומען אן אויטאמאטישע באשטעטיגונג-בליצבריוו. דער ערשטער טייל, דער טריגער, איז שוין כמעט פארטיג. דארט דארף מען אויסקלייבן פון וועלכן Google Sheets טאבעלע מ'נעמט די דאטא.

אין דעם Sheets-פייל זענען דא עטליכע בלעטער. דעריבער דארף מען אויסקלייבן ביי "list" דעם ערשטן בלאט, מיטן נאמען "Form responses 1". דארט לאנדן די ענטפערס פונעם פארם. אזוי פארשטייט n8n וועלכע קאלומען דא זענען און פון וועלכע קאלומען מען דארף ארויסציען די דאטא.

דערנאך שטעלט מען אויך איין דעם "trigger mode" און מען דרוקט צו ציען (fetch) אן event. דער רעזולטאט איז "no data found". דאס איז פונקט וואס מען דארף ערווארטן, ווייל דער פארם איז נאך נישט פארעפנטלעכט און קיינער האט נאך גארנישט אריינגעשריבן. דאס איז געווען נאר א פראבע פון דער פארבינדונג. די פארבינדונג איז געלונגען, און n8n פארשטייט וואס עס געפינט זיך אויפן בלאט.

איינשאלטן די נייטיגע APIs אין Google Cloud (21:58:39)

איצט גייט מען צוריק צו Google Cloud Console, ווייל מען דארף איינשאלטן (enable) דריי APIs: Gmail, Drive און Sheets. די שריט:

- מען קליקט אויף "APIs and services".
- מען זוכט "Google Drive API", מען קליקט דערויף און מען דרוקט "Enable". עס נעמט א מאמענט ביז עס ווערט אקטיווירט.
- דערנאך מעקט מען אויס דאס ווארט "Drive" אינעם זוך-פעלד און מען שרייבט אנשטאט דעם "Sheets". מען קליקט אויף "Google Sheets API" און מען דרוקט ווידער "Enable".
- צולעצט מען שרייבט אנשטאט "Sheets" דאס ווארט "mail" אדער "Gmail". דער ערשטער רעזולטאט איז "Gmail API", און מען שאלט אים אויך איין.

ווען אלע דריי APIs זענען איינגעשאלטן, איז די ארבעט אין Google Cloud Console פארטיג, און דער טריגער איז כמעט אינגאנצן צוגעגרייט.

פארעפנטלעכן דעם פארם און אריינשרייבן פראבע-דאטא (22:00:11)

דער נעקסטער שריט איז פארעפנטלעכן (publish) דעם Google Form וואס מען האט פריער געשאפן. מען גיט א נאמען, און ביי "manage" שטעלט מען איין אז יעדער וואס האט דעם לינק זאל עס קענען עפענען. דערנאך דרוקט מען "publish". מען קאפירט דעם לינק און מען עפנט אים אין א נייעם טאב.

עס ווערט געוויזן א טעות: "requested URL not found". מען פרובירט צו פארשטיין פארוואס. מען קאנטראלירט אז דער פארם איז טאקע פארעפנטלעכט און צוטריטלעך פאר יעדן מיטן לינק. עס ווערט באמערקט אז "anyone with link" דארף געווענליך נישט האבן רעדאקטאר-רעכטן. נאר פאר פראבע-צוועקן לייגט מען צו אן אימעיל-אדרעס וואס זאל האבן צוטריט. ענדלעך שטעלט זיך ארויס אז מען האט קאפירט דעם אומריכטיגן לינק. דער ריכטיגער לינק איז דער פון "share link". מיט אים עפנט זיך דער פארם אן קיין פראבלעם.

אין דעם פארם שרייבט מען אריין:

- א נאמען.
- אן אימעיל-אדרעס, אויסגעקליבן פון די פארשלאגן.
- א טעלעפאן-נומער. מען נעמט א צופעליגן נומער 9123456789, ווייל דער נומער מוז האבן 10 ציפערן.
- א דאטום פון דעם קאלענדאר-פארמאט. מען קלייבט אויס דעם 27סטן פאר דעם אפוינטמענט.

מען צייכנט אויך אן "send a copy of responses" און מען שיקט אוועק. דער ענטפער ווערט פארשריבן. מען באקומט א קאפיע פון די ענטפערס, אבער נאך נישט קיין באשטעטיגונג-בליצבריוו. פאר דעם דארף מען צולייגן א צווייטן טייל צום ארבעטספלוס: אן "action".

קאנטראלירן די דאטא אין n8n (22:02:15)

אין Google Sheets זעט מען שוין אז די אריינגעשריבענע אינפארמאציע איז פארשריבן: דער אדרעס, דער נאמען, דער טעלעפאן-נומער און אזוי ווייטער. צוריק אין n8n, דרוקט מען ווידער fetch און מען זעט דעם רעזולטאט אין דריי פארמען:

- אין טאבעלע-פארעם (table).
- אלס "schema", וואו מען קען זען די דאטא-טיפן פון יעדן פעלד.
- אין JSON פארמאט.

דאס איז פונקט ווי מען וויל אז די דאטא זאל אויסזען. דער טריגער איז אינגאנצן פארטיג.

צולייגן די Gmail אקציע (22:02:46)

מען קליקט צו צולייגן אן נייע אקציע און מען זוכט "Gmail". Gmail האט אסאך מעגלעכקייטן: "add label to message", "delete message", "get message", "mark", "reply to a message", "send message" און נאך. דא וויל מען נאר שיקן א בריוו, דעריבער קלייבט מען אויס "send message".

עס עפנט זיך א פענצטער וואו מען דארף צולייגן א "credential" און איינשטעלן פון וועלכן אדרעס מען שיקט און וואס פאר א מעלדונג מען שיקט. צולייגן דעם credential איז פשוט, ווייל מען האט שוין פריער געשאפן OAuth. מען דרוקט "sign in with Google", מען קלייבט אויס דעם אימעיל מיט וועלכן מען האט געשאפן דעם פראיעקט און די OAuth credentials, און מען גיט רשות.

די איינשטעלונגען פון דער אקציע:

- Resource: message. Operation: send.
- To: דעם אימעיל-אדרעס פונעם באניצער. מען שלעפט אים פשוט אריין (drag and drop) פון די דאטא-פעלדער פונעם טריגער.

- Subject: א טעקסט נאך באליבן. מען שרייבט אריין "Acknowledgement regarding your upcoming appointment".
- Email type: HTML.

שרייבן די מעלדונג און פראבירן (22:04:59)

די מעלדונג הייבט זיך אן מיט "Hi" און דערנאך דעם נאמען פונעם מענטש, וואס מען שלעפט אריין פון די דאטא. אינטערעסאנט איז אז אינעם אונטערשטן טייל פון פענצטער זעט מען שוין גלייך דעם רעזולטאט, א פארויסבליק ווי דער בריוו וועט אויסזען.

ווייטער שרייבט מען בערך אזוי: "We are pleased to inform you that your appointment has been confirmed as per your requested date", און דארט שלעפט מען אריין דעם דאטום-פעלד. דערנאך: "Please be available at least 10 minutes before the schedule", "We hope you have a great day", און צום סוף אן אונטערשריפט ווי "Thanks and regards, Simplilearn".

דערנאך דרוקט מען "Execute step" פאר א פראבע. עס ווייזט זיך "Node executed successfully". אינעם Gmail קאסטן איז אנגעקומען די באשטעטיגונג "0 minutes ago". דער בריוו זאגט אז דער אפוינטמענט ביי Simplilearn איז באשטעטיגט פאר דעם געבעטענעם דאטום, דעם 27סטן פעברואר, און מען זאל קומען לכל הפחות 10 מינוט פריער.

צולייגן אן IF באדינגונג פאר פארגאנגענע דאטעס (22:06:31)

דער קורס הייבט אויף א פראבלעם: וואס אויב א מענטש קלייבט אויס א דאטום וואס איז שוין פארביי? די אויטאמאציע וואלט אים געשיקט א באשטעטיגונג, און דאס טאר נישט פאסירן. די לייזונג איז צולייגן א באדינגונג וואס קאנטראלירט צי דער דאטום איז היינט אדער אין דער צוקונפט. דאס טוט מען מיט אן "If" נאוד:

- אויב דער דאטום איז אין דער צוקונפט (true), גייט מען ווייטער צום נאוד "send an email".
- אויב דער דאטום איז פון דער פארגאנגענהייט, פארבינדט מען צו דעם נאוד "Do nothing", און מען איגנארירט עס.

דאס איז אן פארלענגערונג פונעם זעלבן ארבעטספלוס: מען שטעלט דעם If נאוד גלייך נאכן טריגער און מען גייט ווייטער פון דארט. עס ווערט אויך דערמאנט אז מען קען צופארבינדן א קאלענדאר צו דעם ארבעטספלוס. אזוי קען מען קאנטראלירן אין רעאלע צייט וועלכע צייטן און דאטעס זענען פריי, און די סיסטעם וועט זיין אינגאנצן אפעריבאר.

מוסטערן (templates) אין n8n און אפהיטן דעם ארבעטספלוס (22:08:03)

צום סוף ווייזט מען אז אויפן n8n וועבזייטל, אונטער "workflow automation", זענען דא פארטיגע מוסטערן לויט קאטעגאריעס: AI, Sales, IT Operations, Marketing, Document Ops, Support, און אנדערע. אויב מען זוכט עפעס ספעציפיש, למשל "data analytics", קען מען עס איינשרייבן אינעם זוך-פעלד און באקומען אלע פאסיגע מוסטערן. מען קען קליקן אויף יעדן מוסטער און אנהייבן ארבעטן דערמיט לויט די אייגענע באדערפענישן.

מען היט אפ דעם ארבעטספלוס אונטערן נאמען "my workflow" און מען לאדט אים אראפ, כדי אים צו צושטעלן אינעם באשרייבונג פונעם ווידעא. דערמיט ענדיגט זיך דער טייל.

וואס איז AWS Bedrock? (22:09:04)

א נייער טייל הייבט זיך אן. AWS Bedrock איז אן אינגאנצן פארוואלטעטע (fully managed AI) סערוויס וואס העלפט פראגראמירער בויען און ניצן AI אפליקאציעס, אן זיך דארפן זארגן וועגן איינשטעלן אדער פירן קאמפליצירטע אינפראסטרוקטור. עס גיט צוטריט צו פאר-טריינירטע מאדעלן פון גרויסע פירמעס ווי Amazon, Anthropic Claude, Meta, Mistral AI און אנדערע.

מיט Bedrock קען מען שאפן chatbots, טעקסט-געניראטארן, צוזאמפאסונג-געצייג און אנדערע AI פונקציעס, אן טריינירן מאדעלן פון אנפאנג. עס דערלויבט אויך:

- צופאסן AI מאדעלן מיט אייגענע דאטא.
- צולייגן guardrails, כדי די ענטפערס זאלן זיין זיכער און פאסיג.
- צופארבינדן knowledge bases, כדי דער AI זאל געבן מער געניוע ענטפערס.

ווייל Bedrock איז א טייל פון AWS, ארבעט עס גלאט מיט אנדערע AWS סערוויסעס, און עס איז גרינג אריינצובערענגען AI אין עקזיסטירנדע אפליקאציעס. צי מען וויל אויטאמאטיזירן קונדן-שטיצע, שאפן אינהאלט אדער פארבעסערן באשלוס-מאכן, גיט Bedrock א פשוטן און ווערטשאפטלעך-אויסברייטבארן (scalable) וועג צו ברענגען AI אין א געשעפט.

סערווערלעס אגענטישע ארבעטס-פלוסן אין Amazon Bedrock (22:10:07)

וואס זענען סערווערלעס אגענטישע ארבעטס-פלוסן (22:10:07)

דער קורס הייבט אן א נייעם טעמע: סערווערלעס (serverless) אגענטישע ארבעטס-פלוסן. עס ווערט ערקלערט אז ווי AI-געטריבענע אויטאמאציע אנטוויקלט זיך ווייטער, דארפן ביזנעסער קלוגע אגענטן וואס קענען פארשטיין א באניצער'ס פראגע, ארויסנעמען די פאסיגע אינפארמאציע, און אויספירן באדייטפולע אקציעס. Amazon Bedrock ברענגט אריין סערווערלעס אגענטישע ארבעטס-פלוסן, וואס לאזן ארגאניזאציעס בויען AI-געשטיצטע הילפס-אגענטן וואס רעדן מיט באניצער, ברענגען דאטא אין רעאלער צייט און פירן אויס ביזנעס-ארבעט, אן דארפן פארוואלטן קיין אינפראסטרוקטור.

מען שטעלט עס אקעגן א טראדיציאנעלן AI-טשאטבאט, וואס ענטפערט געווענליך לויט פריער-צוגעגרייטע ענטפערס אדער סטאטישע וויסן. מיט סערווערלעס אגענטישע ארבעטס-פלוסן אין Amazon Bedrock קענען AI-אגענטן דינאמיש ארויסנעמען אינפארמאציע און אויסלייזן אקציעס, וואס מאכט זיי פיל שטארקער ווי פשוטע טשאטבאטס.

די טריט ווען א באניצער רעדט מיט א Bedrock-אגענט (22:10:50)

ווען א באניצער רעדט מיט אן Amazon Bedrock אגענט, פאסירן די פאלגנדע טריט:

- ערשטנס, באארבעטן דעם באניצער'ס איינגאב: דער אגענט באקומט א פראגע, למשל: "איך דארף צוריקגעבן א לעפטאפ וואס איך האב געקויפט פאריגע וואך, קענסטו מיר העלפן?" דער foundation model באארבעט דעם איינגאב, ציט ארויס די וויכטיגע פרטים און באשליסט וואס דער קומענדיגער שריט איז.
- צווייטנס, ארויסנעמען אינפארמאציע פון א knowledge base: אויב מען דארף נאך פרטים, פרעגט דער אגענט א וויסנס-באזע כדי צו ברענגען די פאסיגע אינפארמאציע.

• דריטנס, אויספירן אן אקציע דורך AWS Lambda: אויב דער אגענט דארף טון אן אקציע, רופט ער אן AWS Lambda פונקציע וואס רעדט מיט די הינטערגרונט-סיסטעמען. למשל, די פונקציע ברענגט די פרטים פון דער באשטעלונג, באשטעטיגט צי עס איז בארעכטיגט, און באארבעט די צוריקגעב-בקשה.

• פערטנס, צוזאמשטעלן און שיקן אן ענטפער: דער אגענט זאמלט די אינפארמאציע און בויט אן ענטפער אין נאטירלעכער שפראך פאר'ן באניצער.

• לעצטנס, המשך פון דער סעסיע: אויב דער באניצער פרעגט נאך א פראגע, פירט דער אגענט ווייטער דעם שמועס און האלט דעם קאנטעקסט.

דער גאנצער ארבעטס-פלוס לויפט אויף AWS אינפראסטרוקטור, אזוי אז ביזנעסער קענען ארויסשטעלן AI-הילפס-אגענטן אן זארגן וועגן פארוואלטן סערווערס, וואקסן די רעסורסן, אדער באהאנדלען הינטערגרונט-אויספירונג.

די הויפט-קאמפאנענטן (22:11:58)

• Amazon Bedrock Agent – דער קערן-AI-אגענט וואס רעדט מיט באניצער, פארשטייט זייערע פראגעס און באשליסט וואס איז דער בעסטער וועג צו טון. ער באניצט זיך מיט פריער-טריינירטע foundation מאדעלן פון פארזארגער ווי Amazon, Mistral א.א.וו. צו באארבעטן און שאפן ענטפערס.

• Foundation מאדעלן – גרויסע שפראך-מאדעלן (LLMs) וואס גיבן כח פאר'ן אגענט'ס נאטירלעכע-שפראך פעאיגקייטן; זיי העלפן פארשטיין פראגעס, ארויסנעמען אינפארמאציע און שאפן ענטפערס.

• Action groups – זיי באשטימען וואס דער אגענט קען טון אויסער ענטפערן פראגעס: אויספירן פונקציעס, ברענגען דאטא און רעדן מיט APIs. ביישפילן: ארויסנעמען קונה-באשטעלונגען פון א דאטא-באזע, באארבעטן צוריקצאלונגען דורך אן e-commerce סיסטעם, אדער שאפן סופּפּארט-טיקעטס אין א help desk פלאטפארמע. יעדע action group איז פארבונדן מיט אן AWS Lambda פונקציע, וואס טוט די פאקטישע ארבעט.

• AWS Lambda פאר ביזנעס-לאגיק – וויבאלד Bedrock אגענטן זענען סערווערלעס, ווערט AWS Lambda געוויצט צו אויספירן הינטערגרונט-לאגיק אן פארוואלטן אינפראסטרוקטור. למשל, איין Lambda פונקציע קען נאכקוקן באשטעלונג-היסטאריע, באשטעטיגן בארעכטיגונג און אנהייבן א צוריקגעב-פראצעס; אן אנדערע קען קאנטראלירן דעם סטאטוס פון סופּפּארט-טיקעטס און געבן דערהיינטיגונגען.

• Knowledge bases – העלפן דעם אגענט ארויסנעמען פאקטישע אינפארמאציע פון סטרוקטורירטע דאטא-באזעס, FAQs אדער דאקומענטן.

• Guardrails – פאר AI-זיכערהייט און נאכפאלגן כללים; זיי זארגן אז דעם אגענט'ס ענטפערס בלייבן זיכער, עטיש און פאסיג, דורך אויסזיפן שעדלעכן, סענסיטיוון אדער פארבאטענעם אינהאלט.

באניצונגען אין דער פראקטיק (22:13:41)

• קונה-סערוויס אויטאמאציע: א פירמע קען ארויסשטעלן א Bedrock -געשטיצטן אגענט וואס ענטפערט FAQs דורך א knowledge base, ברענגט באשטעלונג-פרטים דורך אן action group מיט א Lambda פונקציע, און הייבט אן צוריקגעב-בקשות און באארבעט צוריקצאלונגען אויטאמאטיש.

• IT help desk: אן אינערלעכער AI-הילפער קען העלפן ארבעטער לייזן טעכנישע פראבלעמען – קאנטראלירן סערווער-uptime דורך AWS Lambda, צוריקשטעלן פאסווערטער דורך אן action group, אדער אריינגעבן IT-טיקעטס אויטאמאטיש.

- HR הילפער: א Bedrock אגענט קען העלפן מיט HR-פראגעס, ווי ברענגען וויפיל פריי-טעג מען האט נאך, באארבעטן פריי-טעג בקשות, אדער געבן פרטים פון פירמע-פאליסיס.

אין קורצן, ווערט געזאגט, עפענען סערווערלעס אגענטישע ארבעטס-פלוסן אין Amazon Bedrock א שטארקן, וואקסבארן און ביליגן וועג צו בויען AI-אויטאמאציע – אגענטן וואס זענען אויסגעריכט אויף אקציעס און גייען ווייטער ווי א פשוטער שמועס. אין די קומענדיגע קאפיטלען וועט מען זען ווי אזוי צו שאפן אן אגענט מיט Amazon Bedrock, ווי אזוי צו פארבינדן מיט א CRM און מאכן חשבונות, ווי אזוי צו לייענען אן FAQ-העלפבוך, און א קורצער דורכגאנג פון דער Amazon Bedrock קאנסאל.

דער מאדעל און זיין פרייז (22:14:58)

דער קורס באניצט זיך מיט'ן מאדעל Claude 3 Haiku, איינער פון די נייע AI-מאדעלן. דער פרייז: $0.25 פאר 1 מיליאן איינגאב-טאקענס, און $1.25 פאר 1 מיליאן אויסגאב-טאקענס. דאס וועט זיין דער הויפט-מאדעל דורכאויס דעם קורס.

ערשטער קאפיטל: צוגרייטן די סביבה (22:15:28)

אין ערשטן קאפיטל בויט מען א פשוטן קלאוד-באזירטן אגענט אין Amazon Bedrock, און מען לערנט ווי אזוי אים אנצורופן (invoke) און נאכפאלגן זיין ארבעטס-פלוס דורך traces, כדי צו זען קלאר וואס עס פאסירט.

דער ערשטער שריט, ווי אין יעדן קאפיטל, איז אימפארטירן די ביבליאטעק boto3 – AWS'ס אפיציעלער SDK פאר Python. אבער פריער דארף מען צוגרייטן די סביבה: א שטיקל קאוד וואס באנוצט) os.environלמשל פאר'ן agent role) לייגט צוריק די סביבה און גרייט צו די רעסורסן. ביים לויפן איז ערשט ארויסגעקומען א syntax error, וואס מ'האט פארראכט; דערנאך זעט מען די מעלדונגען אז דער agent reset, Lambda reset און guardrail reset זענען פארענדיגט – זאכן וואס מ'האט שוין דורכגערעדט אין אריינפיר-קאפיטל.

שאפן א client און דעם אגענט (22:17:15)

נאך'ן אימפארטירן boto3 שאפט מען א client-אביעקט כדי צו רעדן מיט די פארשידענע AWS סערוויסעס:

boto3.client(service_name='bedrock-agent', region_name='us-west-2') = bedrock_agent

מיט דעם client פארבינדט מען זיך צו AWS און שטעלט דארט אויף דעם אגענט. עס ווערט באטאנט אז דער קאוד קאנפיגורירט א קלאוד-אגענט וואס ארבעט אויף פראדוקציע-שטאפל; ער לויפט נישט אינעם notebook אליין.

צו שאפן דעם אגענט רופט מען די פונקציע create_agent פונעם bedrock_agent client, מיט די פאראמעטערס:

- נאמען – נאר פאר אונזער רעפערענץ; דער אגענט אליין דערקענט אים נישט. דא: "laptop customer support agent".
- Foundation model – אויסקלייבן א גרויסן שפראך-מאדעל פון Bedrock וואס וועט באהאנדלען די שפראך-פארשטאנד. מען לייגט אריין דעם model ID פון Anthropic Claude 3 Haiku – א קליינער, שנעלער און זייער פעאיגער מאדעל. די פולע ליסטע מאדעלן געפינט זיך אין AWS'ס דאקומענטאציע, וואס מ'וועט ווייזן אין שפעטערדיגע קאפיטלען.

- Instructions – אויך געוראפן system prompts, וואס באשטימען דעם אגענט'ס אויפפירונג. דא: "You are an advanced AI agent acting as a frontline customer support agent" – א פראנטליין קונה-סערוויס אגענט, ספעציעל פאר לעפטאפ-ביזנעסער.
- Resource role – אן IAM role וואס איז שוין פריער געשאפן אין דער סביבה; מען נעמט זיין ARN (Amazon Resource Name) פון אן environment variable. די role באשטימט די זיכערהייט-ערלויבענישן, ד.ה. צו וועלכע סערוויסעס דער אגענט האט צוטריט. מען האלט עס מינימאל לויטן פרינציפ פון least privilege – דער אגענט האט צוטריט נאר צום נייטיגן שפראך-מאדעל און גארנישט מער.

דער רעזולטאט ווערט געשפייכלט אין א וואריאבל create_agent_response, און ביים נאכקוקן זעט מען אז דער אגענט איז געשאפן. מען באמערקט אז מ'האט נאך נישט צוגעטיילט קיין געצייג; אין Bedrock הייסן זיי actions און געהערן צו action groups. דערווייל איז עס א באזיס-AI לעפטאפ-סופפארט אגענט; דער קומענדיגער שריט איז אים אנרופן און רעדן מיט אים.

די שטאפלען פונעם אגענט און הילפס-פונקציעס (22:21:20)

אין דער ענטפער זעט מען דעם סטאטוס "creating", דעם agent ID און דעם נאמען. דער אגענט גייט דורך דריי שטאפלען:

- Creating – ער ווערט אויפגעשטעלט (דאס פאסירט יעצט).
- Not prepared – ער עקזיסטירט, אבער איז נאך נישט גרייט.
- Prepared – גרייט צום באנוצן.

ווען ער איז prepared קען מען שאפן אן alias – א סטאבילע ווערסיע פונעם אגענט פאר פראדוקציע. אנשטאט טוישן דעם אגענט גלייך, קאנטראלירט מען דערמיט די דערהיינטיגונגען, פארוואלט פארשידענע ווערסיעס און האלט אלעס סטאביל. אבער פריער מוז מען ווארטן ביז ער דערגרייכט "not prepared".

מען שפייכלט דעם agent ID אין א וואריאבל, גענומען פון create_agent_response אונטער'ן פעלד agentId → . דערנאך שטעלט מען אויף הילפס-פונקציעס, וואס העלפן ווארטן אויף געוויסע פראצעסן (ווי דער איבערגאנג צו prepared), כדי מען זאל נישט דארפן אלעס מאכן מיט דער האנט. עס ווערט אויך דערמאנט אז פאר לעפטאפ-סופפארט וועלן אזעלכע פונקציעס מאכן גרינגער ארבעט ווי נאכקוקן די warranty, לייזן hardware-פראבלעמען און פירן קונים דורך software-תיקונים.

מען אימפארטירט פון helper די פונקציע wait_for_agent_status, און רופט זי מיט'ן agent ID און target status "NOT_PREPARED". מען זעט אז דער אגענט איז אריבער פון creating צו not prepared – פונקט וואס מ'האט געוואלט.

צוגרייטן דעם אגענט (22:24:20)

דער קומענדיגער שריט איז צוגרייטן דעם אגענט או ער זאל זיין גרייט. מען רופט די פונקציע prepare_agent פונעם bedrock_agent client און גיט איבער דעם agent ID פונעם וואריאבל, כדי צו זאגן מיט וועלכן אגענט מען ארבעט. דער סטאטוס ווערט "preparing". דער ציל איז "prepared", דעריבער נעמט מען ווידער דעם wait_for_agent_status קאוד, דאס מאל מיט'ן target status "PREPARED".

אגענט-אליאס, שמועס מיט'ן אגענט און אקציעס (22:25:30)

שאפן אן אליאס פארן Bedrock אגענט (22:25:30)

דער אגענט האט שוין דערגרייכט דעם "prepared" צושטאנד. מען זעט אז פאר דעם האט ער אויך א קורצע ווייל געווען "preparing", און ערשט דערנאך איז ער אריבער צו "prepared".

דער נעקסטער שריט איז שאפן אן אליאס (alias). דער קורס ערקלערט פארוואס: מען וויל רופן (invoke) דעם אגענט און פירן מיט אים א שמועס. ווייל דער אגענט איז שוין גרייט, קען מען יעצט אויפשטעלן דעם אליאס דורך דעם Bedrock agent קליענט.

פארן שאפן דעם אליאס דארף מען געבן עטליכע וויכטיגע פרטים:

- דעם agent ID, כדי די סיסטעם זאל וויסן מיט וועלכן אגענט מען ארבעט.
- א נאמען פארן אליאס. דער נאמען קען זיין וואס מען וויל. פאר פשטות רופט מען אים "my agent alias".

מען רופט די פונקציע create_agent_alias, און מען גיט איר ביידע, דעם agent ID און דעם אליאס-נאמען. דעם ענטפער פון דער אפעראציע לייגט מען אריין אין א וואריאבל, כדי מען זאל שפעטער קענען צוריקקומען צו אים וועו מען דארף.

פארן לויפן דעם קאוד איז דא נאך א שריט, כדי אלעס זאל גיין גלאט. מען נעמט ארויס פונעם ענטפער דעם agent alias ID און מען היט אים אפ אין א באזונדערן וואריאבל. דאס איז וויכטיג, ווייל דורך אים וועט מען נאכגיין דעם סטאטוס פונעם אליאס.

כדי צו זיין זיכער אז דער אליאס איז ריכטיג אויפגעשטעלט, ניצט מען א helper פונקציע. די פונקציע וואלט ביז דער סטאטוס פונעם אליאס ווערט "prepared". אזעלכע helper פונקציעס זענען שוין פריער ערקלערט געווארן.

וועו מען לויפט דעם קאוד, ארבעט אלעס ווי געהעריג. דער אליאס גייט אריבער אין "prepared" צושטאנד, פונקט ווי מען האט געווארט. מיט א גרייטן אליאס איז דער אגענט שוין פונקציאנעל, דאס הייסט אז מען קען שוין רעדן מיט אים.

רופן דעם אגענט און א session ID (22:29:00)

כדי צו רעדן מיט'ן אגענט דארף מען שאפן א נייעם קליענט, ספעציעל פאר דעם צווועק. מען שאפט אים מיט boto3, אזוי ווי פריער. מען גיט אן דעם service name און דעם region, און מען לייגט דעם קליענט אריין אין א וואריאבל. דאס איז א bedrock agent runtime קליענט.

דערנאך קען מען שוין טאקע רופן דעם אגענט:

- ערשט אימפארטירט מען uuid.
- דערנאך רופט מען די מעטאד invoke_agent פונעם bedrock agent runtime קליענט. דאס איז כמעט אלעס וואס מען דארף כדי אנצוהייבן א שמועס מיט'ן אגענט.

די מעלדונג וואס מען שיקט, ווי א קונה, איז: "העלא, איך האב נעכטן געקויפט א לעפטאפ אין אייער געשעפט, עס איז דא א פראבלעם מיט'ן דיספליי און איך וויל אים צוריקגעבן."

אין דעם רוף גיט מען אן:

- דעם agent ID, כדי צו זאגן וועלכער אגענט עס איז.
- דעם agent alias ID וואס מען האט פריער אפגעהיטן.
- די מעלדונג אליין.

דער קורס באמערקט אז אין דעם שטאדיום האט דער אגענט נאך נישט קיין אקציעס קאנפיגורירט. פונדעסטוועגן דארף ער נאך אלץ ארויסגעבן א ענטפער לויט דער לאגיק וואס איז יא פארהאן.

אחוץ דער מעלדונג גיט מען אויך א session ID, כדי אויפצוהאלטן דעם שמועס. דער קורס ברענגט א משל: ווער עס האט אמאל געשמועסט מיט א customer support אגענט, האט באמערקט אז יעדער שמועס האט א session פאר זיך. ווען מען ענדיגט א שמועס און מען הייבט אן א נייעם, איז דאס א באזונדערע נייע session. דאס זעלבע דארף מען דא שאפן א session ID כדי צו האלטן דעם שמועס.

דעם יחידישן ID שאפט מען מיט Python'ס uuid מאדול. דערפאר האט מען עס פון אנפאנג אימפארטירט. אזוי איז יעדע session יחידיש צו דערקענען. די היסטאריע פונעם שמועס ווערט אפגעהיטן אינעם קלאוד, אזוי אז מען דארף זי נישט אליין פירן.

נאך צוויי סעטינגס:

- endSession = False. דאס האלט דעם שמועס אקטיוו, אנשטאט אים צו פארלירן נאך איין איבערשמועס.
- enableTrace איז דערווייל False. דאס שאלט אפ דעם tracing פון די ריקוועסטס. מען וועט עס שפעטער אויספארשן.

ארויסנעמען דעם ענטפער פונעם event stream (22:32:05)

נאכן לויפן דעם קאוד און דרוקן דעם ענטפער, זעט מען אז ער ווערט נישט געוויזן ווי א געוועלכער טעקסט. דער ענטפער קומט צוריק ווי אן event stream, נישט ווי א פשוטע מעלדונג. מען קען אים דעריבער נישט גלייך דרוקן. מען דארף ארויסציען דעם אמת'ן ענטפער פונעם stream, כדי צו זען וואס דער אגענט האט טאקע געזאגט.

מען שרייבט אזוי: event_stream איז גלייך צו דעם "completion" טייל פונעם invoke_agent ענטפער. דערנאך לויפט מען איבער יעדן event אינעם stream און מען דרוקט אים.

מען זעט אז עס קומט צוריק נאר איין הויפט-event, וואס הייסט "chunk". אינעווייניג ליגט דאטא, ספעציעל bytes, מיט'ן ענטפער פונעם אגענט. דער אגענט האט געענטפערט: "עס טוט מיר באנג, אבער איך קען אייך דערווייל נישט העלפן מיט'ן צוריקגעבן דעם לעפטאפ. ביטע ווענדט זיך דירעקט צו אונזער customer support גרופע, זיי וועלן אייך קענען העלפן."

דער קורס ערקלערט אז דער ענטפער מאכט סענס. דער אגענט האט נאך נישט קיין אקציעס. ער וויסט אז ער דארף האבן געוויסע פונקציעס, אבער ווייל מען האט קיינע נישט דעפינירט, באקענט ער פשוט אז ער האט באגרענעצונגען.

עפענען דעם trace-מאוד (22:33:38)

עס ווערט געפרעגט: ווי פארשטייט דער אגענט דאס אלעס, און ווי טוט ער די נויטיגע אקציע? כדי צו זען א קלארן בילד דארף מען אנצינדן דעם trace-מאוד. פריער האט מען אים אפגעשלאסן מיט False, יעצט שטעלט

מען אים אויף True. דאס הייבט אן א נייע session, עס שטעלט אלעס צוריק, און עס לאזט פירן נאך א שמועס מיט'ן אגענט. דאס מאל זעט מען מער ווי אזוי ער שטעלט צוזאם זיין ענטפער.

נאכן אנצינדן דעם trace און עס דרוקן, זעט מען באקאנטע פרטים, ווי דעם agent ID, דעם session ID, און נאך אסאך אנדערע. דער סוף-ענטפער לויטעט: "כדי אנצוהייבן דעם פראצעס פון צוריקגעבן אייער לעפטאפ מיט'ן דיספליי, ביטע זאמלט די פאלגנדע אינפארמאציע: דעם דאטום פונעם קויף, דעם אָרדער-נומער אדער קבלה, און א באשרייבונג פונעם פראבלעם."

דער קורס שטעלט צוזאם די צוויי ענטפערס. דאס ערשטע מאל האט דער אגענט געזאגט אז ער קען נישט העלפן, און מען זאל זיך וועגדן צו א מענטשלעכן support אגענט. יעצט האט ער טאקע געגעבן א ענטפער: ער בעט פרטים, ווי דעם דאטום פונעם קויף און דעם אָרדער-נומער אדער קבלה, כדי מען זאל קענען אנהייבן דעם צוריקגעבן-פראצעס.

דער נעקסטער שריט איז צוגעבן אקציעס צום אגענט, כדי ער זאל קענען טאן נוצליכע ארבעט. דאס וועט קומען אינעם קומענדיגן קאפיטל.

נייער קאפיטל: פארבינדן דעם אגענט מיט דרויסנדיגע סערוויסעס (22:35:11)

יעצט וואס מען האט א פשוטן אגענט וואס לויפט, גייט מען א שריט ווייטער. אין דעם קאפיטל פארבינדט מען דעם אגענט מיט דרויסנדיגע סערוויסעס, אזוי אז ער זאל קענען ארבעטן מיט דער דרויסנדיגער וועלט. מען וועט סימולירן א פארבינדונג מיט א CRM סיסטעם, וואו דער אגענט קען ארויסנעמען פרטים וועגן קונים און פארשרייבן support tickets. דאס מאכט דעם customer support אגענט מער פונקציאנעל און נענטער צו א ריכטיגן פראקטישן באניץ.

די שריט:

- אויפשטעלן די סביבה (environment). דאס נעמט א שטיק צייט.
- אימפארטירן די נויטיגע לייבראריס: boto3, כדי צו ארבעטן מיט AWS סערוויסעס, דעם uuid מאדול פונעם פריערדיגן קאפיטל פאר יחידישע session IDs, און עטליכע helper פונקציעס פון helper.

ווייל די environment variables זענען שוין אויפגעשטעלט, איז מען גרייט. דער אגענט פונעם פריערדיגן קאפיטל לויפט נאך, און מען קען אים ווייטער ניצן. דאס מאל גיט מען אים מער פרטימדיגע אנווייזונגען, ווייל מען האט צוגעלייגט עטליכע עקסטערע טיילן.

מען שאפט א נייעם session ID און מען שטעלט אויף א נייע מעלדונג, ענלעך צו די פריערדיגע. דערנאך ניצט מען א helper פונקציע, invoke_agent_and_print. זי שיקט די מעלדונג און דרוקט דעם ענטפער. אין איר גיט מען אן דעם agent ID, דעם session ID, דעם טעקסט פון דער מעלדונג, און דעם agent alias ID.

דער ענטפער זעט אויס ווי א טשאט. מען האט געשריבן: "מיין נאמען איז Leah, מיין לעפטאפ האט פראבלעמען מיט'ן דיספליי, און איך וויל א refund." דער אגענט האט געענטפערט אז עס טוט אים באנג, אבער ער האט נישט גענוג אינפארמאציע כדי דורכצופירן דעם refund, און ער האט אויסגערעכנט די פרטים וואס ער דארף.

Action groups און AWS Lambda (22:39:00)

יעצט דארף מען פארבינדן דעם אגענט מיט אנדערע געצייג, כדי ער זאל זיין מער נוצליך. אין Amazon Bedrock agents רופן זיך די געצייג "actions" און "action groups". זיי לאזן דעם אגענט טאן ארבעט וואס גייט ווייטער ווי בלויז ענטפערן מיט טעקסט.

די לאגיק פון די אקציעס ווערט געפירט דורך AWS Lambda פונקציעס. די Lambda פונקציע איז דער פונקט וואוהין דער אגענט שיקט ריקוועסטס ווען מען דארף טאן אן אקציע. ווען דער אגענט רופט די פונקציע, שיקט ער איר אן event מיט וויכטיגע פרטים: דעם נאמען פונעם אגענט, דעם נאמען פונעם action group, די ספעציפישע פונקציע, און סיי וועלכע פאראמעטערס.

כדי צו פארבינדן דעם אגענט מיט געצייג, דעפינירט מען אן action group מיט'ן bedrock agent קליענט. דער action group לאזט דעם אגענט טאן ארבעט דורך א AWS Lambda פונקציע.

ערשט שאפט מען דעם קליענט, אזוי ווי פריער. דער service name איז bedrock-agent, און דעם region גיט מען דערווייל נישט אן. דערנאך שטעלט מען אויף דעם action group:

- מען גיט דעם action group א נאמען: "customer support actions". דער נאמען איז בלויז פאר אונזער אייגענער רעפערענץ.
- מען פארבינדט אים מיט'ן אגענט דורך דעם agent ID פון די environment variables.
- מען גיט אן די Lambda פונקציע וואס פירט די לאגיק, דורך איר ARN.
- מען דעפינירט די פונקציעס וואס דער אגענט קען רופן.

די ערשטע פונקציע איז customerId lookup. זי זוכט ארויס א קונה'ס ID לויט פרטים ווי email, נאמען אדער טעלעפאן-נומער. דער אגענט מוז שיקן כאטש איינס פון די פרטים כדי ווייטער צו גיין. די פאראמעטערס זענען באשריבן אין נאטירלעכער שפראך, כדי דער אגענט זאל פארשטיין זייער צוועק.

די צווייטע פונקציע איז sendToSupport. זי שיקט דעם ענין ארויף צו א גרופע מענטשלעכע support ארבעטער, למשל ווען דער אגענט וואס מען האט געשאפן קען נישט ענטפערן. זי פארלאנגט סיי א customer ID און סיי א support summary. אויב דער אגענט האט נישט קיין customer ID, נעמט ער אים ערשט ארויס דורך דער customerId lookup פונקציע.

אגענט-טעסטן, גארדרעילס און קאוד-אינטערפרעטער אין Bedrock (22:41:36)

ענדיגן דעם action group און ווארטן אויפן ריכטיגן סטאטוס (22:41:36)

דער קורס ערקלערט אז איידער מען שיקט אוועק דעם רעקוועסט, ווערט אנגעגעבן אז די agent version איז draft. דאס מיינט אז די ענדערונג גייט אריין אין דעם יעצטיגן אויפשטעל פונעם אגענט. ווען אלעס איז גרייט, ווערט אויסגעפירט דער רעקוועסט, און דער רעזולטאט ווערט געהאלטן אין א ווערייבל וואס הייסט create agent action group response.

מען קוקט אויפן רעזולטאט. דער רעקוועסט איז אויסגעפירט געווארן, און דער ענטפער אנטהאלט א action group ID, ווי אויך דעם action group state, וואס איז enabled.

אין דער לעקציע ווערט באטאנט אז מען דארף זיכער מאכן אז דער action group איז אינגאנצן enabled איידער מען גייט ווייטער. אין דעם פאל איז ער שוין enabled. פונדעסטוועגן לייגט מען צו א helper פונקציע וואס ווארט ביז דער סטאטוס איז ריכטיג. דער צוועק: ווען מען לויפט דעם גאנצן סקריפט פון אנהייב ביז סוף, זאל אלעס גיין גלאט, אן קיין פראבלעמען פון טיימינג.

די טריט זענען:

- מען נעמט ארויס דעם action group ID פונעם ענטפער, כדי מען זאל זיך שפעטער קענען באציען צו אים.
- מען רופט די helper פונקציע, פונקט ווי פריער אין קורס, מיט דעם target status אויף enabled.
- היות מען האט געמאכט אן ענדערונג אינעם אגענט, דארף מען אים צוגרייטן נאכאמאל (prepare), כדי די ענדערונגען זאלן אריינגענעמען. דערנאך דערהיַנטיקט מען דעם alias, אז ער זאל ווייזן אויף דער לעצטער ווערסיע. מען זעט אז דער אגענט איז אריבער אין prepared state.
- מען דערהיַנטיקט דעם alias, און אויך ער דערגרייכט דעם prepared state . דערפאר ניצט מען די prepare agent פונקציע.

יעצט איז דער alias אינגאנצן דערהיַנטיקט, און דער אגענט איז גרייט צו באנוצן.

טעסטן דעם אגענט מיט אן אימעיל-אדרעס (22:44:11)

יעצט קען מען דורכפירן ארבעטן און טעסטן דעם אגענט מיט א מעסעדזש. קודם שטעלט מען צוריק (reset) דעם session ID. דערנאך גיט מען א נייעם מעסעדזש, וואס אנטהאלט דאס מאל אן אימעיל-אדרעס: "My name is Nia", מיט א פאלשן אימעיל וואס ענדיגט זיך מיט leia.com, "My laptop has display issues, and I want a refund". דאס מיינט: דער לעפטאפ האט פראבלעמען מיטן סקרין, און מען וויל צוריק דאס געלט.

דער קורס ערקלערט אז מיט דעם אימעיל קען דער אגענט מיט זיכערקייט צופאסן דעם באניצער צו א customer ID. אין דעם פאל וועט דאס זיין סתם א ראנדאם נומער. דאס סימולירט אן אמת'ע אינטעראקציע פון קאסטומער-סופּפּארט.

מען שרייבט א helper פונקציע וואס רופט invoke agent, שיקט דעם מעסעדזש און דרוקט אויס דעם ענטפער. מען גיט איבער דעם agent ID, מיט קאפי-פעיסט, ווייל די פונקציע איז שוין פריער געגוצט געווארן.

אין ערשטן לויף איז enable trace געשטעלט אויף false. מען זעט די פראגע פונעם באניצער און דעם ענטפער פונעם אגענט: "Hello Leah, I'm sorry to hear you are having issues with your laptop display". דער אגענט בעט מער פרטים וועגן דעם ספעציפישן פראבלעם.

עס ווערט ערקלערט אז אפשר האט דער אגענט שוין באקומען דעם customer ID דורכן אימעיל, און דערפאר פרעגט ער יעצט אנדערע פרטים: וואס פונקטליך איז דער פראבלעם, ווען איז געקויפט געווארן, און אזוי ווייטער. צום סוף זאגט ער אז מען זאל אים לאזן וויסן, און ער וועט העלפן אריבערגעבן (escalate) דעם ענין צום קאסטומער-סופּפּארט טים, אויב נייטיק.

דער "דענק-פראצעס" פונעם אגענט מיט trace (22:46:54)

דערנאך שטעלט מען enable trace אויף true. יעצט זעט מען נישט נאר די פראגע פונעם באניצער און דעם לעצטן ענטפער, נאר אויך דעם thought process פונעם אגענט, דאס הייסט וואס גייט פאר אין אים אינעווייניג.

דער אגענט שטעלט צוזאמען די פרטים וואס ער האט שוין: דעם נאמען פונעם קאסטומער, דעם אימעיל, דעם פראבלעם (דער לעפטאפ האט display issues) און די בקשה (א refund). ער קומט צום אויספיר אז ער פעלט נאך וויכטיגע פרטים כדי אינגאנצן צו באארבעטן דעם refund, למשל דער order number אדער די דאטע וועו דער לעפטאפ איז געקויפט געווארן. אן די פרטים קען ער נישט אריבערגעבן דעם פאל צום סופּפּארט טים. דערפאר באשליסט ער צו פרעגן דעם באניצער, און דאס איז פונקט וואס ער האט געטון.

צוזאמגעפאסט: ביז יעצט האט מען געבויט אן אגענט וואס ענטפערט עפעקטיוו און ניצט tools צו באארבעטן קאסטומער-סופּפּארט פעלער. ער קען ארויסנעמען א customer ID און פארשרייבן (log) א סופּפּארט רעקוועסט, און דאדורך איז ער אסאך מער פונקציאנעל.

וואס זענען גארדרעילס אין Amazon Bedrock (22:47:57)

דער נעקסטער טשעפטער באהאנדלט guardrails אין AWS Bedrock. עס ווערט ערקלערט אז guardrails אין Amazon Bedrock זענען א סעט זיכערהייט-מיטלען וואס זארגן אז די ענטפערס וואס AI שאפט זאלן זיין זיכער, עטיש און פאסיג. זיי ארבעטן ווי פילטערס און באגרעניצונגען, וואס לאזן נישט דעם AI אגענט ארויסגעבן שעדליכן, פארטייאישן (biased) אדער פארבאטענעם אינהאלט.

guardrails זענען וויכטיג צו רעגולירן ווי AI פירט זיך, בפרט אין קאסטומער-סופּפּארט, געזונטהייט (healthcare), פינאנצן און אנדערע סענסיטיוו פעלדער, וואו די ענטפערס מוזן האלטן ביי די compliance סטאנדארטן.

ווי ארבעטן זיי? וועו א AI מאדעל שאפט אן ענטפער, אנאליזירן די guardrails דעם רעזולטאט איידער ער גייט צום באניצער. אויב דער ענטפער אנטהאלט פארבאטענעם, באליידיגנדיגן אדער פאליסי-ברעכנדיגן אינהאלט, בלאקירן אדער ענדערן אים די guardrails. מען קען זיי צופאסן לויט די באדערפענישן פונעם ביזנעס. די הויפט-אייגנשאפטן זענען:

- content filters: זיי סקענען און פילטערן די AI ענטפערס פאר גראבע וועדטער, hate speech אדער באליידיגנדיגן אינהאלט, און בלאקירן זיי איידער זיי דערגרייכן דעם באניצער.
- denied topics: פריער באשטימטע טעמעס וואס דער AI טאר נישט באהאנדלען, למשל פאליטישע מיינונגען, פערזענליכע געזונט-עצות, און רעקאמענדאציעס פאר פינאנציעלע אינוועסטמענטס.
- harmful categories: זיכערהייט-סעטינגס וואס מען קען צופאסן, צו באגרעניצן ענטפערס וואס האבן צו טון מיט שנאה, זיך-שאדן, געוואלד, harassment און אומאנשטענדיגן אינהאלט.
- custom word filtering: באניצער קענען באשטימען א ליסטע פון פארבאטענע ווערטער אדער פראזן וואס דער AI טאר נישט ארויסגעבן.
- custom guardrails via AWS Lambda: פאר מער פארגעשריטענע פעלער קען מען שרייבן אייגענע לאגיק מיט AWS Lambda, וואס פרופט און ענדערט די AI ענטפערס דינאמיש.

פיר טריט צו איינפירן גארדרעילס (22:50:28)

דער קורס ווייזט, מיט קאוד-ביישפילן, פיר טריט:

- טריט 1, שאפן א guardrail: מען ניצט דעם AWS SDK פאר Python, boto3, כדי צו באשטימען content filtering, פארבאטענע טעמעס און סענסיטיוויטעט-לעוועלס. דער קאוד שאפט א guardrail מיטן נאמען customer support guardrail . ער פילטערט שעדליכע שפראך, בלאקירט ספעציפישע ווערטער (למשל

fraud, scam, illegal), באגרעניצט שמועסן אויף געוויסע טעמעס ווי פאליטיק, רעליגיע און אינוועסטמענטס, און שטעלט אן פארשידענע זיכערהייט-לעוועלס פאר פארשידענע קאטעגאריעס אינהאלט.

• טריט 2, צובינדן דעם guardrail צו א AI אגענט: נאכדעם וואס דער guardrail איז געשאפן, מוז מען אים צוטשעפען צו אן Amazon Bedrock אגענט, כדי ער זאל טאקע דורכפירן דאס פילטערן אין די ענטפערס. דאס זארגט אז דער אגענט פאלגט די כללים און גיט נישט ארויס פארבאטענעם אינהאלט.

• טריט 3, טעסטן דעם guardrail דורכן רופן דעם אגענט: מען שיקט א ביישפיל-פראגע און קוקט אויפן ענטפער. אויב דער guardrail איז ריכטיג אנגעווענדט, וועט דער אגענט אפזאגן צו געבן אינווסטמענט-עצות, און ענטפערן מיט א זיכערן, פריער-באשטימטן ענטפער, אזויווי: "I'm sorry, but I cannot provide financial investment advice".

• טריט 4, פיין-צופאסן דעם guardrail: אויב דער אגענט גיט נאך אלץ אומגעוואונטשענע ענטפערס, קען מען צופאסן דעם guardrail דורכן ענדערן די סענסיטיוויטעט-לעוועלס אדער צולייגן נייע denied topics. אין דעם ביישפיל-קאוד וועט דער אגענט נישט ענטפערן אויף פינאנציעלע אדער מעדיצינישע פראגעס, כדי צו האלטן ביי די רעגולאציעס.

פארוואס זענען גארדרעילס אזוי וויכטיג (22:52:02)

עס ווערן אויסגערעכנט פיר סיבות:

• צו פארמיידן מיסברויך פון AI: זיי באגרעניצן די ענטפערס און לאזן נישט שעדליכע, פארטייאישע אדער אומעטישע ענטפערס.

• צו האלטן ביי רעגולאציעס: אינדוסטריעס ווי פינאנצן, געזונטהייט און לעגאלע סערוויסעס מוזן האלטן ביי שטרענגע כללים, און guardrails לאזן נישט דעם AI געבן עצות אן רשות.

• צו פארגרעסערן דאס צוטרוי פון קאסטומערס: א גוט-אויסגעשטעלטער AI אסיסטענט היט אפ דעם שם פונעם בראנד, ווייל אלע ענטפערס שטימען מיט די פאליסיס פון דער פירמע און מיט עטישע כללים.

• צו פארמינערן AI ריזיקעס: זיי דינען ווי א זיכערהייט-נעץ קעגן פאלשע אינפארמאציע, hate speech אדער באליידיגנדיגן אינהאלט.

צוזאמגעפאסט: guardrails אין Amazon Bedrock זענען א קריטישע אייגנשאפט, כדי AI מאדעלן זאלן ארויסגעבן זיכערע, עטישע ענטפערס וואס שטימען מיט די פאליסיס. דורך content filters, denied topics און באגרעניצונגען פון harmful categories קענען ביזנעסער קאנטראלירן וואס דער AI גיט ארויס און באשיצן די אינטעראקציעס מיט באניצער. מיט AWS Lambda און צופאסבארע פילטערס קענען פראגראמירער נאך מער צופאסן די guardrails לויט די ביזנעס-באדערפענישן. צי עס גייט אין פארמיידן אז AI זאל געבן פינאנציעלע עצות, אין בלאקירן באליידיגנדיגע שפראך, אדער אין האלטן קאסטומער-סופּפּארט קאנסיסטענט: guardrails העלפן האלטן AI פארלעסליך און צוטרוי-ווערדיג אין אמת'ע אפליקאציעס.

א קאוד-אינטערפרעטער פארן אגענט: אויפשטעלן די סביבה (22:53:33)

דער נעקסטער טשעפטער מאכט דעם אגענט קלוגער: אנשטאט גלייך צו שיקן א בקשה צום סופּפּארט, וועט ער קודם דורכפירן חשבונות. מען גיט אים א code interpreter. דאס מיינט אז דער אגענט וועט קענען מאכן גענויע חשבונות דורכן שרייבן און לויפן צייטווייליגן Python קאוד, כדי צו שטיצן זיינע ענטפערס.

ווי אין אלע טשעפטערס, לויפט מען קודם דעם קאוד וואס שטעלט אויף די סביבה. מען לאדנט אלע נייטיגע ביבליאטעקן, ווי פריער: boto3 און די helper פונקציעס. דערנאך שאפט מען א bedrock agent client, אויך ווי פריער.

דער אגענט אין דער סביבה איז דער זעלבער ווי אין לעצטן טשעפטער. כדי צו פארגרעסערן זיינע פעאיגקייטן, דערהיַנטיקט מען אים און לייגט צו נייע אייגנשאפטן.

דערהיַנטיקן דעם action group מיט א נייער פונקציע (22:55:14)

דער קאוד דערהיַנטיקט אן עקזיסטירנדן action group אין Amazon Bedrock: ער ענדערט דעם function schema און לייגט צו א נייע פונקציע. דער קורס ערקלערט די טיילן:

- די פונקציע update agent action group ענדערט דעם action group פונעם אגענט, כדי ער זאל קענען דורכפירן די נייטיגע אקציעס ביים באהאנדלען קאסטומער-סופּפּארט בקשות.
- action group name איז customer support action, דער זעלבער נאמען ווי פריער.
- action group state ווערט קלאר געשטעלט אויף enabled. ביים שאפן איז אן action group ממילא enabled, אבער ביים דערהיַנטיקן מוז מען דאס אליין אנגעבן, כדי ער זאל בלייבן אקטיוו.
- מען גיט איבער דעם action group ID און דעם agent ID, און די agent version איז draft. Bedrock ערלויבט עטליכע ווערסיעס פונעם אגענט, און ענדערונגען גייען קודם אריין אין draft, איידער זיי ווערן פיינאל.
- lambda function arn: דער אגענט פירט אויס די אקציעס דורך א Lambda פונקציע, און איר ARN ווערט געהאלטן אינעם ווריאבל lambda function arn.
- function schema: דא זענען דא דריי פונקציעס. די ערשטע איז די customer ID פונקציע, וואס העלפט דעם אגענט אידענטיפיצירן א קאסטומער לויט די פרטים וואס זענען פאראן. איר באשרייבונג זאגט בערך: נעם א customer ID לויט די פאראנענע פרטים; מען מוז שיקן כאטש איין פאראמעטער; דאס איז פריוואטע אינפארמאציע און טאר נישט ווערן געגעבן צום באניצער. דאס מיינט אז דער אגענט קען אויפזוכן דעם customer ID, אבער טאר אים נישט ווייזן דעם באניצער. די פאראמעטערס קענען זיין email, name אדער phone number, און כאטש איינער פון זיי איז פארלאנגט.

אגענט מיט Code Interpreter און דאקומענטן (22:57:46)

די נייע פונקציע purchase search (22:57:46)

אין דער פריערדיגער לעקציע האט דער אגענט שוין געהאט צוויי פונקציעס: customer ID, וואס זוכט אויס דעם קונה, און send to support. send to support ווערט גענוצט ווען דער אגענט קען נישט אליין לייזן דעם פראבלעם, און דאן גיט ער איבער די בקשה צו א מענטשליכן אגענט. יעצט ווערט צוגעלייגט א נייע פונקציע וואס הייסט purchase search.

אין דער באשרייבונג פון דער פונקציע שטייט אז זי זוכט א קויף וואס דער קונה האט געמאכט און ברענגט אירע פרטים. די פרטים קען מען נוצן ווען מען עפנט א סופּאָרט-בקשה. דער אגענט מעג באשטעטיקן אז ער האט די דאטא, למשל "איך האב געטראפן דעם קויף" אדער "איך קען נישט טרעפן דעם קויף". אבער פריוואטע אינפארמאציע, ווי אָרדער-נומערן, טאר ער נישט ארויסגעבן.

די פונקציע באקומט דריי פאראמעטערס:

- customer ID, די נומער פונעם קונה;
- product description, א באשרייבונג פונעם פראדוקט;
- purchase date, דער טאג פונעם קויף. ער מוז זיין אין פארמאט יאר-חודש-טאג.

דער ציל פון דער פונקציע איז אז דער אגענט זאל קענען פרובירן אויב דער קויף איז טאקע געווען, איידער ער שיקט דעם פאל ווייטער. אזוי ווייסט מען אז די טענה איז אמת'דיג. דער גאנצער ענטפער ווערט אפגעהיט אין א וואריאבל וואס הייסט update agent action group response. אזוי בלייבן די אלטע פונקציעס (customer ID און send to support) און די נייע פונקציע ווערט צוגעלייגט.

ארויסנעמען דעם action group ID און א נייע פראבלעם מיט דאטומען (22:59:19)

נאכן לויפן דעם קאוד נעמט מען ארויס דעם action group ID פון דעם ענטפער. דער ענטפער פון דער פונקציע update agent action group איז געבויט אין שיכטן, און דער ID ליגט אינעווייניג אין action group. א שורה קאוד נעמט אים ארויס און לייגט אים אריין אין א וואריאבל action group ID, כדי מען זאל אים קענען נוצן שפעטער. מען זעט אז דער סטאטוס פון דער action group איז enabled.

דערנאך שיקט מען א נייעם מעסעדזש. ער הייבט זיך אן מיטן אימעיל פונעם קונה און זאגט ווייטער: "איך האב געקויפט א לעפטאפ מיט פינף וואכן צוריק, יעצט האט ער א פראבלעם מיטן סקרין, איך וויל א refund." דא קומט ארויס א פראבלעם. די פונקציע פארלאנגט דעם דאטום אין פארמאט יאר-חודש-טאג, אבער דער באניצער האט געשריבן "מיט פינף וואכן צוריק". דעריבער דארף מען דערהיינטיקן דעם אגענט און אים צולייגן א code interpreter, וואס זאל קענען ארבעטן מיט דאטומען.

צולייגן א code interpreter action group (23:01:28)

דער קאוד וואס קומט יעצט לאזט דעם אגענט אויספירן Python -קאוד ביים לויפן. ער טוט עס דורך שאפן א code interpreter action group. דער פראצעס האט דריי הויפט-טריט.

דער ערשטער טריט איז שאפן די action group. מען רופט די פונקציע create agent action group און שאפט א נייע action group מיטן נאמען code interpreter action. א action group באשטימט א סעט ארבעטן וואס דער אגענט קען טון. דא גיט זי דעם אגענט די מעגלעכקייט צו לויפן Python -קאוד דורך דער Amazon Bedrock code interpreter אייגנשאפט.

די וויכטיגע פאראמעטערס זענען:

- action group name;
- action group status;
- agent ID;
- agent version, וואס איז נאך אלץ draft;
- parent action group signature, מיטן ווערט Amazon code interpreter.

דער לעצטער פאראמעטער פארבינדט די action group מיט דעם פארטיג-געבויטן code interpreter פון Amazon Bedrock. אזוי קען דער אגענט גלייך לויפן Python-קאוד, למשל פאר חשבונות, אנאליז פון דאטא און אויטאמאציע. דאס איז וויכטיג, ווייל Amazon Bedrock שטעלט צו אזעלכע איינגעבויטע action groups

פאר געווענליכע AI-ארבעטן. וועו מען ווייזט אויף Amazon code interpreter, קען דער אגענט פראצעסירן און לויפן Python-קאוד אן דעם וואס מען זאל דארפן שרייבן אייגענע lambda-פונקציעס.

דער צווייטער טריט איז ארויסנעמען דעם ID פון דער נייער action group. דער ID ווייזט אן אויף דער code interpreter action group און ווערט אפגעהיט אין code interpreter action group ID . דערנאך קומט א helper-פונקציע, וואס האלט אפ דאס לויפן ביז די action group איז אינגאנצן גרייט.

צוגרייטן דעם אגענט און דערהיינטיקן דעם alias (23:03:01)

דער דריטער טריט איז צוגרייטן (prepare) דעם אגענט און דעם alias, כדי די נייע action group זאל אריינגיין. ערשט שרייבט מען prepared agent response, מיטן agent ID. דערנאך קאפירט מען די helper-פונקציע מיטן target state "prepared" און לויפט דעם קאוד. עס קומט ארויס א טעות, וואס ווערט פאררייכט. דערנאך זעט מען אז דער קאוד ווארט ביז דער סטאטוס פונעם אגענט ווערט prepared, און צום סוף קומט ער אן צו dem סטאטוס.

נאכדעם דערהיינטיקט מען דעם alias מיט update agent alias. מען גיט אריין agent ID, דעם agent alias ID, און דעם agent alias name "test". מען קלעבט אריין די helper -פונקציע און לויפט. מען זעט אז דער alias איז אויך דערהיינטיקט גע wוארן און איז אנגעקומען צום סטאטוס prepared.

פרובירן דעם אגענט מיטן חשבון פון דאטום (23:06:06)

מען שטעלט א נייעם session ID און גיט אריין דעם זעלבן מעסעדזש. דערנאך שרייבט מען קאוד וואס דרוקט אויס דעם ענטפער, דורך דער פונקציע invoke agent and print. זי באקומט agent ID, agent alias ID, input text (דער מעסעדזש) און session ID. enable trace ווערט געשטעלט אויף true. די פונקציע שיקט דעם מעסעדזש צום אגענט און ברענגט צוריק דעם ענטפער.

אינעם ענטפער זעט מען דעם "agent's thought process", דעם invocation input , נאכאמאל דעם thought process, און צום סוף דעם final response. דער ענטפער זאגט: "איך האב אדורכגעקוקט אייער בקשה און געזען אז איר האט געקויפט דעם לעפטאפ בערך מיט פינף וואכן צוריק. ווייל דאס איז אין דער געווענליכער צייט פאר א refund, האב איך איבערגעגעבן אייער בקשה צום סופּאָרט-טיעם. זיי וועלן אדורכקוקן די פרטים און זיך באלד מעלדן." עס ווערט אויך געגעבן א נומער פאר דעם support case, ווי א טיקעט.

אינעם thought process זעט מען ווי דער אגענט האט געטראכט. די purchase search פונקציע האט געטראפן א קויף פאר דעם קונה וואס פאסט צו די פרטים. דאן דארף דער אגענט אויסרעכענען דעם דאטום פונעם קויף, כדי צו וויסן צי ער איז אין דער refund-צייט, ווייל מען האט אים נישט געגעבן דעם ריכטיגן דאטום, נאר "פינף וואכן צוריק". דער אגענט האט גענוצט Python צו רעכענען וועלכער דאטום איז געווען מיט פינף וואכן צוריק. ערשט דערנאך איז ער געקומען צום שלוס אז דער קויף איז אין דער refund-צייט.

באקומען דעם ענטפער אין streaming פארמאט (23:09:00)

דערנאך פרובירט מען נאכאמאל, מיט א נייעם session ID און דעם זעלבן מעסעדזש. דאס מאל וויל מען באקומען דעם ענטפער אין streaming פארמאט. מען שטעלט אויף דעם client און רופט invoke agent response. דערנאך שאפט מען אן event stream פון דעם completion פונעם ענטפער.

א לופּ גייט איבער יעדן event אין dem event stream און קוקט צי ער אנטהאלט א chunk. אויב יא, איז די דאטא געהאלטן אין bytes, און מען מוז זי decode'ן צו א לייענבארן string. דער טעקסט ווערט פארמאטירט ווי JSON און אויסגעדרוקט. אנדערע סארטן events, ווי metadata אדער events פון פונקציעס, ווערן אויסגעדרוקט ווי זיי זענען, פאר אנאליז. דאס tracing ווייזט ווי דער אגענט האט באהאנדלט די בקשה, און דער שמועס בלייבט אפן, אזוי אז מען קען ווייטער רעדן.

דער output איז זייער לאנג, און עס ווערט נאר געגעבן א תמצית. ער איז א trace log פון דעם orchestration פראצעס פון Amazon Bedrock agents ביים באהאנדלען דעם פאל. מען זעט די גאנצע היסטאריע: דער אגענט שפילט די ראלע פון א frontline customer support, און ער האט באקומען א בקשה פון א קונה מיטן נאמען Leah. זי האט געמעלדט א פראבלעם מיטן סקרין פון א לעפטאפ, געקויפט מיט פינף וואכן צוריק.

דער trace ווייזט דעם אגענט'ס חשבון שריט ביי שריט, און וועלכע פונקציעס ער האט גערופן:

- ערשט האט ער אויסגעזוכט דעם customer ID לויטן אימעיל;
- דערנאך האט ער געזוכט די פרטים פונעם קויף מיט purchase search;
- צום סוף האט ער איבערגעגעבן דעם פאל צום סופּאָרט-טיעם מיט send to support.

דער אגענט האלט זיך שטרענג ביי די כללים, כדי ער זאל זיין גענוי. ער נוצט דעם Python REPL פאר חשבונות מיט דאטומען, און guardrails לאזן אים נישט מאכן פאלשע הנחות. אינעם output זענען אויך דא רופן צום מאדעל, רופן צו פונקציעס, API-ענטפערס און trace logs, וואס באשטעטיקן אז די בקשה איז מצליח באהאנדלט געווארן. דער לעצטער ענטפער לאזט וויסן: "איך האב אדורכגעקוקט Leah'ס קויף-פרטים און איבערגעגעבן איר refund-בקשה צום סופּאָרט-טיעם. זיי וועלן זיך באלד מעלדן צו איר צו העלפן מיטן סקרין-פראבלעם. לאזט מיך וויסן אויב איר דארפט נאך הילף." דער לעצטער טייל פונעם output באשטעטיקט אז דער ענטפער איז פארטיג און איבערגעגעבן געווארן.

נייער קאפיטל: פארבינדן דעם אגענט מיט סופּאָרט-דאקומענטן (23:13:37)

אין דעם נייעם קאפיטל ווערט געוויזן ווי אזוי צו פארבינדן דעם אגענט מיט א repository פון customer support דאקומענטן. די דאקומענטן באשרייבן פשוטע פראבלעמען וואס דער אגענט קען אליין לייזן. ווען ער איז פארבונדן, קען ער לייזן געוויסע פראבלעמען גלייך, און די איבעריגע איבערגעגעבן צו א מענטשליכן workflow.

ערשט לויפט מען דעם קאוד וואס שטעלט אויף די סביבה. דערנאך אימפארטירט מען boto3, JSON און די helper-פונקציעס. מען שאפט א client מיט boto3.client, מיט service name "bedrock-agent" און region name "us-west-2".

דערנאך שרייבט מען קאוד וואס ברענגט די פרטים פונעם אגענט וואס עקזיסטירט שוין אין Amazon Bedrock. מען רופט get agent מיטן agent ID, און דער ענטפער ווערט אפגעהיט אין describe agent response. די פונקציע get agent ברענגט metadata און קאנפיגוראציע-פרטים פון דעם ספעציפישן Bedrock agent.

דערנאך דרוקט מען אויס דעם ענטפער:

print(json.dumps(describe_agent_response, indent=4, default=str))

אזוי ווערן די פרטים פונעם אגענט אויסגעדרוקט אין א געארדנטן פארמאט וואס א מענטש קען גרינג לייענען. json.dumps פארוואנדלט דעם dictionary אין describe agent response אין א string אין JSON פארמאט. API-ענטפערס קומען אפט מיט דאטא-סטרוקטורן געבויט אין שיכטן, און א גוטער פארמאט מאכט עס גרינגער צו אנאליזירן דעם output און געפינען טעותים. דער פאראמעטער indent=4 גיט דעם JSON א קלארע אויסשטעלונג מיט ריכטיגע אפשטאנדן. אן indentation וואלט דער ענטפער ארויסגעקומען אלס איין לאנגע שורה, וואס איז שווער צו לייענען.

צוגעבן א וויסן-באזע און דער Bedrock קאנסאל (23:17:42)

אריסדרוקן דעם אגענט'ס אנווייזונגען (23:17:42)

דער קורס האלט ביים ארויסדרוקן די ענטפער פון דער פונקציע וואס באשרייבט דעם אגענט. עס ווערט ערקלערט אז די ענטפער איז שווער צו ליינען מיט די אויגן. דערפאר נוצט מען json.dumps, און מען שטעלט אריין default=str. דאס איז כדי צו באהאנדלען אביעקטן וואס מען קען נישט גלייך איבערמאכן אויף JSON, למשל צייט-שטעמפלען (timestamps) אדער ספעציעלע דאטא-טיפן. אזעלכע אביעקטן ווערן פארוואנדלט אין טעקסט (strings) איידער מען דרוקט זיי ארויס. אזוי שטעלט מען זיך פאר טעותים וואס קענען פאסירן ווען די ענטפער אנטהאלט קאמפליצירטע דאטא-סטרוקטורן.

ווען מען לאזט לויפן דעם באפעל, זעט מען קלאר דעם גאנצן אויפשטעל פונעם אגענט:

- דער נאמען
- דער סטאטוס
- די ווערסיע
- די פארבינדענע "action groups"
- די איינשטעלונגען פאר'ן אויספירן

עס ווערט באטאנט אז די אויסשריפט איז נוצלעך פאר עטליכע זאכן. מען קען זיך פארזיכערן אז דער אגענט איז ריכטיג אויפגעשטעלט. מען קען קוקן אין וועלכן צושטאנד ער געפינט זיך איידער מען מאכט ענדערונגען. און מען קען זוכן פראבלעמען אין דעם אויפשטעל. ווייל אלע פרטים ווערן געווויזן אויף אן ארגאניזירטן אופן, ווערט דאס "דיבאגן" גרינגער.

מען זעט אז צווישן די אינפארמאציע געפינען זיך אויך די אנווייזונגען (instructions). דערפאר שרייבט מען נאך א שורה קאוד וואס דרוקט זיי באזונדער ארויס:

```
print(describe_agent_response['agent']['instruction'])
```

עס ווערט ערקלערט אז דער דיקשאנערי describe_agent_response אנטהאלט פארשידענע "מעטא-דאטא" וועגן דעם אגענט. דער שליסל agent האלט די הויפט-פרטים. דאס פעלד instruction האלט דעם סיסטעם-פראמפט, דאס הייסט די פארויס-באשטימטע אנווייזונג וואס זאגט דעם אגענט ווי זיך צו פירן און ווי צו ענטפערן אויף פראגעס.

וואס שטייט אין די אנווייזונגען (23:19:13)

די ארויסגעדרוקטע אנווייזונג הייבט זיך אן מיט: "דו ביסט א פראנטליין קאסטומער-סופּפּארט אגענט פאר א פירמע, און דיין ראלע איז צו באהאנדלען קונים' מעלדונגען". נאכדעם קומען כללים ווי אזוי צו באהאנדלען א קונה'ס מעלדונג:

- ערשט אנאליזירן די מעלדונג כדי צו פארשטיין וואס דער פראבלעם איז.
- באשטימען וואס איז די פאסיגע אקציע.
- נוצן די פאסיגע געצייג (tools).
- געבן א קלארע און קורצע צוזאמענפאסונג פון דער אנאליז.

דערצו איז דא א טייל וועגן באהאנדלען פראבלעמען לויט ווי קאמפליצירט זיי זענען. עס זענען דא צוויי סארטן. די ערשטע זענען קאמפליצירטע פראבלעמען אדער אזעלכע וואס פאדערן א מענטשליכע אריינמישונג. די צווייטע זענען פשוט'ע פראבלעמען, ווי אלגעמיינע פראגעס ווי אזוי צו נוצן דעם פראדוקט.

דער קורס ווייזט אן אז אויב מען גייט דורך די אויסשריפט, זעט מען אז עס פעלט א וויסן-באזע (knowledge base). מען האט געגעבן אנווייזונגען און כללים, אבער קיין וויסן-באזע איז נישטא. דערפאר איז דער קומענדיגער שריט צוצוגעבן איינע.

ארויסנעמען די וויסן-באזע (23:20:52)

מען שרייבט קאוד וואס רופט אויף די פונקציע get_knowledge_base מיט דעם פאראמעטער knowledgeBaseId גלייך צו דעם knowledge_base_id. עס ווערט ערקלערט אז דער קאוד נעמט ארויס אינפארמאציע וועגן א וויסן-באזע וואס איז פארבונדן מיט דעם Amazon Bedrock אגענט.

א וויסן-באזע איז א זאמלונג פון אפגעהיטענע אינפארמאציע, למשל) FAQsאפט-געפרעגטע פראגעס) אדער די פירמע'ס כללים. דער אגענט קען דאס נוצן כדי צו געבן בעסערע ענטפערס. דער ID פארזיכערט אז מען קומט צו צו דער ריכטיגער וויסן-באזע.

די ענטפער ווערט אפגעהיטן אין get_knowledge_base_response. זי אנטהאלט פרטים ווי דער נאמען פון דער וויסן-באזע, דער סטאטוס, די באשרייבונג און די אפגעהיטענע דאקומענטן. ווען מען לאזט לויפן די פונקציע, קען מען זיך איבערצייגן אז די וויסן-באזע עקזיסטירט, אז זי איז אקטיוו, און אז זי איז ריכטיג פארבונדן מיטן אגענט. דאס איז נוצלעך ווען דער אגענט דארף ארויסנעמען אינפארמאציע אבער ענטפערט נישט ווי געהעריג.

דערנאך דרוקט מען די ענטפער ארויס מיט json.dumps, און מען זעט אז זי אנטהאלט א סך אויפשטעל-פרטים.

פארבינדן די וויסן-באזע מיטן אגענט (23:23:02)

איצט פארבינדט מען די וויסן-באזע מיט דעם אגענט דורך דער פונקציע associate_agent_knowledge_base. מען גיט אירː

- דעם agentId
- דעם knowledgeBaseId
- agentVersion גלייך צו "DRAFT"

• א description, "My KB"

עס ווערט ערקלערט אז דער קאוד פארבינדט א וויסן-באזע מיט דעם Amazon Bedrock אגענט. אזוי קען דער אגענט נוצן די אפגעהיטענע אינפארמאציע ווען ער שאפט ענטפערס.

די ווערסיע "DRAFT" פארזיכערט אז די ענדערונג ווערט צוגעוועגדט צום אגענט'ס ענטוואורף-ווערסיע, אידער מען פארענדיגט אים. די באשרייבונג "My KB" גיט א קורצן נאמען פאר דער פארבינדונג.

אזוי קען דער אגענט אריינקוקן אין דער וויסן-באזע און זיך פארלאזן אויף איר ווען ער ענטפערט אויף פראגעס. דאס פארבעסערט די געניקייט און די קוואליטעט פון די ענטפערס. עס איז נוצלעך פאר קאסטומער-סופּפּארט, FAQs, און ארויסנעמען אינפארמאציע פון סטרוקטורירטע דאקומענטן.

אין דער אויסשריפט זעט מען אז די וויסן-באזע איז "ENABLED", און מען איז גרייט ווייטער צו גיין.

צוגרייטן דעם אגענט און דעם אליאס (23:25:39)

דער קוד פאר דעם שריט ווערט אריינגעקלעבט פון פריער, ווייל מען האט דאס שוין געטון עטליכע מאל. עס ווערט נאכאמאל ערקלערט אז ער באשטייט פון פיר שריט:

• prepare_agent: מען רופט עס אויף מיט דעם agentId. דאס זאגט Amazon Bedrock צו באארבעטן און אקטיווירן די לעצטע ענדערונגען אין דעם אגענט, ווי למשל דערהיינטיגטע אקציעס אדער א ניי-פארבונדענע וויסן-באזע.

• ווארטן ביז דער אגענט איז גרייט: ווייל די צוגרייטונג נעמט צייט, פארזיכערט די פונקציע wait_for_agent_status אז דער אגענט קומט אן צום צושטאנד "PREPARED" איידער מען גייט ווייטער. דאס שטעלט זיך פאר טעותים וואס קומען פון אויפרופן א אגענט וואס איז נאך נישט אינגאנצן אקטיוו.

• דערהיינטיגן דעם אליאס: די פונקציע update_agent_alias גיט דעם אגענט א נייעם אליאס, ווי למשל "my agent alias". א אליאס איז א רעפערענץ צו א ווערסיע. ער דערלויבט צוצוקומען צו פארשידענע ווערסיעס פונעם אגענט אויף א סטרוקטורירטן אופן. דער שריט פארזיכערט אז דער אליאס איז ריכטיג פארבונדן מיטן דערהיינטיגטן אגענט.

• ווארטן ביז דער אליאס איז אקטיוו: די פונקציע wait_for_agent_alias_status פארזיכערט אז דער אליאס איז אינגאנצן גרייט איידער מען נוצט אים. אזוי ווען יעדע בקשה צום אגענט ארבעטן ריכטיג.

די שריט פארענדיגן אלע דערהיינטיגונגען און פארזיכערן אז דער אגענט ארבעט אינגאנצן איידער מען רופט אים אויף. אזוי שטעלט מען זיך פאר טעותים, האלבע דערהיינטיגונגען, אדער א אגענט וואס ענטפערט נישט. דער אגענט ווערט פארלעסלעך און גרייט צו באהאנדלען פראגעס. מען לאזט לויפן דעם קאוד און ווארט ביז ער ענדיגט.

אויספרובירן דעם קאסטומער-סופּפּארט אגענט (23:27:23)

נאכן אויספירן האט מען שוין א פולשטענדיג געבויטן קאסטומער-סופּפּארט אגענט. מען שאפט א session ID און א מעלדונג, פונקט ווי פריער. די מעלדונג קומט פון אן אימעיל-אדרעס ביי vr.com: "איך האב געקויפט א לעפטאפ מיט פינף וואכן צוריק און איצט האט ער פראבלעמען מיטן סקרין. איך וויל צוריק געלט."

מען רופט אויף די פונקציע invoke_agent און דרוקט ארויס די ענטפער, ווען די trace איז אויסגעלאשן. דער אגענט ענטפערט בערך אזוי: "איך האב איבערגעקוקט די בקשה פאר צוריקגעלט פאר דעם לעפטאפ וואס איר האט געקויפט מיט בערך 5 וואכן צוריק און האט איצט סקרין-פראבלעמען. איך האב געקענט אויסזוכן די

פרטים פון אייער קויף. לויט מיין אויספארשונג האב איך איבערגעגעבן אייער פאל צו אונזער סופּפּארט-גרופּע, און א סופּפּארט-אגענט וועט זיך פארבינדן מיט אייך."

דערנאך פרובירט מען א צווייטע מעלדונג: "איך קען נישט פארבינדן מיין לעפטאפ צום Wi-Fi, וואס קען איך טון?". דאס מאל איז די trace אנגעצונדן. אין דער אויסשריפט זעט מען אז די guardrail trace איז none.

מען זעט אויך דעם אגענט'ס געדאנקען-גאנג. ערשט קומט: "לאמיך זען ווי איך קען העלפן מיט דעם פראבלעם אז מען קען נישט פארבינדן דעם לעפטאפ צום Wi-Fi". דערנאך גייט ער דורך די ווי‌סן-באזע. צום סוף גיט ער א ענדגילטיגן ענטפער מיט פינף שריט ווי אזוי צו פרובירן פארריכטן דעם פראבלעם. דער קורס מוטיגט צו שטעלן נאך פראגעס און זען ווי דער אגענט ענטפערט.

פארוואס דער אגענט איז שטארק (23:29:58)

עס ווערט ערקלערט אז דער אגענט איז געבויט צו נאכגיין דעם גאנג פונעם שמועס און צו נוצן "אגענטיק" טראכטן (agentic reasoning) כדי צו באשליסן ווי צו באהאנדלען יעדע בקשה. ער ענטפערט נישט בלינדערהייט. ער אנאליזירט די לאגע און טרעפט אויס דעם בעסטן וועג.

זיין כח איז אז ער קען נעמען פארשידענע וועגן לויטן פראבלעם. א פשוט'ן פראבלעם לייזט ער אליין. ווען א לאגע פאדערט מענטשליכע הילף, ווייסט ער ווען צו איבערגעבן דעם פאל צו א סופּפּארט-אגענט.

די באשלוסן זענען אפהענגיג אין דריי זאכן:

- דער אגענטיק ארבעטס-גאנג (workflow) וואס פירט זיינע באשלוסן.
- די אנווייזונגען וואס מען גיט אים כדי צו פארמען זיין אויפפירונג.
- די געצייג און אקציעס וואס ער האט צוטריט צו, ווי ווי‌סן-באזעס און APIs, ווי מען האט געזען אין דעם ביישפיל.

צווזאמען מאכן די דריי עלעמענטן דעם אגענט עפעקטיוו און אנפאסבאר, און ער קען גלאט באהאנדלען פארשידענע סארטן שמועסן מיט קונים.

א קורצער אריינבליק אין דעם Amazon Bedrock קאנסאל (23:30:28)

דער קומענדיגער קאפיטל איז א דורכגאנג פון דעם Amazon Bedrock קאנסאל. עס ווערט ערקלערט אז נאכדעם וואס מען האט געבויט דעם אגענט מיט קאוד, איז וויכטיג צו זען ווי דער קאנסאל גיט א וויזועלן און אינטעראקטיוון וועג. דארט קען מען אויפשטעלן AI מאדעלן, פירן אגענטן, און פיינער אנפאסן פונקציעס ווי guardrails און ווי‌סן-באזעס.

קאוד גיט מער בייגעוודיגקייט און קאנטראל. דער קאנסאל פארגרינגערט אבער דעם פראצעס מיט א באקוועמען אינטערפעיס, וואו מען קען שנעל מאכן ענדערונגען, אויספרובירן און זוכן פראבלעמען.

Amazon Bedrock איז א טייל פון AWS'ס "serverless" AI אינפראסטרוקטור. דערפאר פאסט ער זיך גלאט צוזאם מיט אנדערע AWS סערוויסעס, און גיט באניצער צוטריט צו פריער-טריינירטע AI מאדעלן פון פארשידענע פירמעס. דאס מאכט אים אן אידעאלן געצייג פאר פירמעס און פראגראמירער וואס ווילן בויען AI לייזונגען אן א טיפע מומחיות אין מאשין-לערנען.

כדי אריינצוקומען, דארף מען זיך אריינלאגן אין דעם AWS אקאונט, זוכן "bedrock" אין דער זוך-שורה, און אויסקלייבן Amazon Bedrock. אין דעם לינקן מעניו זעט מען אפציעס פאר פארשידענע טיילן פון Bedrock:

- מאדעל-פראווייִדערס
- פּירן אגענטן
- guardrails
- וויסן-באזעס
- "playgrounds" צו אויספֿרובירן מאדעלן אינטעראקטיוו

דער צענטראלער דעשבאורד דערלויבט גרינג צו פּירן AI ארבעטס-גאנגען, אויפּצושטעלן ווי דער AI געהילף זאל זיך פּירן, און צו פֿארבעסערן די מעגלעכקייט ארויסצונעמען אינפֿארמאציע.

די מאדעלן און ווי מען עפּנט זיי (23:31:59)

איינע פון די וויכטיגסטע זייטן פון Bedrock איז די גרויסע אויסוואל מאדעלן פון פּירנדיגע AI פֿארשונג-ארגאניזאציעס. זיי האבן פֿארשידענע מעגלעכקייטן, ווי שאפן טעקסט, טראכטן (reasoning), צוזאמענפאסן, און ענטפערן אויף פֿראגעס.

אין דעם providers-טייל ווערן זיי אויסגערעכנט, צווישן זיי: Amazon, Anthropic (Claude מאדעלן), AI21 Labs, Cohere, Meta, Mistral AI, Stability און נאך. יעדער פֿראווייִדער גיט אנדערע סארטן גרויסע שפּראך-מאדעלן, וואס מען נוצט לויט די באדערפענישן פונעם פֿראיעקט. למשל, Anthropic'ס Claude מאדעלן זענען באקאנט פֿאר א צוגאנג וואס שטעלט זיכערקייט אויפן ערשטן ארט, און Meta'ס Llama מאדעלן זענען געבויט פֿאר עפּעקטיוו reasoning-ארבעטן.

א ווארענונג: אויב מען באקומט טעותים ווען מען פֿרובירט נוצן געוויסע מאדעלן, איז מסתמא דער טעם אז זיי זענען נאך נישט אקטיווירט פֿאר דעם AWS רעגיאן אדער אקאונט. מען קען זיי אקטיווירן ביי די model access איינשטעלונגען. דארט קען מען:

- מסכים זיין צו די לייסענס-אפּמאכן (end user license agreements) כדי צו אקטיווירן ספּעציפֿישע מאדעלן.
- ענדערן די צוטריט-רשותן כדי Bedrock זאל זיך קענען פֿארבינדן מיט יענע מאדעלן.
- אויסקלייבן אדער אראפּנעמען מאדעלן לויט די באדערפענישן פונעם פֿראיעקט.

ווען א מאדעל איז אקטיווירט, קען מען אים נוצן אין אגענטן, וויסן-באזעס און אנדערע AI אפּליקאציעס. וויטער וועלן זיך דער קורס צו דעם agents-טייל, וואו מען שטעלט אויף און פּירט AI געהילפן. עס ווערט אנגעהויבן צו ערקלערן אז איינע פון די שטארקסטע מעגלעכקייטן פון Bedrock איז צו שאפן AI אגענטן וואס קענען אינטעראקטירן.

Bedrock קאנסאל און דער ערשטער n8n וואָרקפֿלאו (23:34:04)

אגענטן, אקשן-גרופּעס און טעסטן אין Bedrock קאנסאל (23:34:04)

דער קורס ערקלערט אז אגענטן אין Amazon Bedrock זענען געבויט צו אינטעראקטירן מיט באניצער, באהאנדלען פֿראגעס און געבן קלוגע ענטפערס לויט דער דאטא וואס איז פֿאראן. דער "Agents" אפּטייל אין קאנסאל גיט מכשירים צו פֿארוואלטן די אגענטן.

וועו מען עפנט דעם אפטייל זעט מען א ליסטע פון די עקזיסטירנדע אגענטן מיט זייער סטאטוס. ווי מען האט שוין געזען בשעת'ן שרייבן קאוד, קען אן אגענט זיין אין צוויי מצבים:

- Prepared: גרייט צום באנוץ, און ענטפערט אקטיוו אויף פראגעס.
- Not Prepared: דער אגענט איז שוין געשאפן, אבער נאך נישט אינגאנצן קאנפיגורירט.

צו שאפן א נייעם אגענט קליקט מען "Create Agent", און מען גיט אריין א נאמען, א באשרייבונג, און מען קלייבט אויס דעם גרויסן שפראך-מאדעל וואס וועט טרייבן די ענטפערס. עס ווערט באטאנט אז דער שריט איז זייער וויכטיג, ווייל די אויסוואל פונעם מאדעל באשטימט די קוואליטעט, דעם סטיל פון די ענטפערס און די פעאיקייטן פונעם אגענט.

דער "Agent Builder" גיט אן אָרדנטלעכן אינטערפעיס, וואו מען קען:

- אריינשרייבן פרטימדיגע אינסטרוקציעס פארן אגענט;
- אויסקלייבן דעם בעסט-פאסנדן AI מאדעל;
- אנצינדן דעם Code Interpreter, וואס לאזט דעם אגענט לויפן Python סקריפטן וועו עס פעלט אויס;
- צופאסן דעם וואָרקפלאו און די לאגיק לויט די געשעפט-באדערפענישן.

א הויפט-טייל פון אויפשטעלן אן אגענט איז קאנפיגורירן "Action Groups", וואס באשטימען וואס דער אגענט קען טאקע טון. זיי לאזן דעם אגענט זיך פארבינדן מיט דרויסנדיגע מכשירים, דאטא-באזעס און סערוויסעס, און אזוי טון ארבעטן וואס גייען ווייטער פון א פשוטן שמועס. אין קאנסאל פארבינדט מען אגענטן מיט דרויסנדיגע סערוויסעס דורך AWS Lambda פונקציעס. Amazon Bedrock מאכט עס גרינגער מיט דער אפציע "Quick Create Lambda Function", וואס:

- שאפט אויטאמאטיש א מוסטער Lambda קאוד;
- שטעלט אויף די נויטיגע AWS דערלויבענישן צום לויפן;
- לאזט פראגראמירער צופאסן די Lambda פונקציעס פאר ספעציפישע געשעפט-פעלער.

א ביישפיל: אויב א סופָארט-אגענט דארף ארויסנעמען די פרטים פון א קונה'ס באשטעלונג, קען מען אויפשטעלן אן אקשן-גרופע וואס פרעגט א דאטא-באזע דורך א Lambda פונקציע. וועו די פונקציע ווערט אויסגעלעזט, באארבעט דער אגענט דעם ענטפער און גיט אים איבער צום באניצער.

נאכן קאנפיגורירן אן אגענט, גיט דער קאנסאל אן אינטעראקטיוו טעסט-סביבה, וואו מען קען שיקן מוסטער-פראגעס, זוכן פעלערן וועו דער אגענט פירט זיך אומגעריכט, און זיכער מאכן אז די פארבינדונג מיט דרויסנדיגע סיסטעמען ארבעט ריכטיג.

Guardrails: קאנטראלירן און פילטערן די ענטפערס (23:35:52)

דער קורס ערקלערט אז זיכערקייט און האלטן זיך אין די כללים זענען קריטיש וועו מען שטעלט ארויס אויטאמאטישע אסיסטענטן. Amazon Bedrock גיט "Guardrails", וואס מען האט שוין געזען אין קאוד אין פריערדיגן קאפיטל, און וואס לאזן פילטערן און קאנטראלירן די ענטפערס פון אן אגענט, אז ער זאל זיך פירן אין די אויפגעשטעלטע גרעניצן.

דער אונטערשייד: אקשן-גרופעס באשטימען וואס אן אגענט קען טון, און גארדרעילס באשטימען וואס ער טאָר נישט טון. זיי בלאקירן אומפאסיגן, שעדליכן אדער פארבאטענעם אינהאלט. דורך דעם אינטואיטיוון אינטערפעיס פון קאנסאל קען מען:

- אנגעבן "Denied Topics", ענינים וואס דער אגענט דארף אויסמיידן;
- באשטימען אינהאלט-פילטערס צו בלאקירן אומפאסיגע ענטפערס;
- אויסשטעלן די זיכערקייט-שטאפלען מיט אינטעראקטיווע "סליידערס".

א ביישפיל: אויב אן ארגאניזאציע וויל נישט אז דער AI אסיסטענט זאל געבן פינאנציעלע עצות, קען מען אריינשטעלן א פארבאטענעם ענין אין די גארדרעיל-סעטינגס.

Knowledge Bases: פארבריייטערן דאס וויסן פונעם אגענט (23:37:07)

א "Knowledge Base" שטארקט די פעאיקייט פון אן AI אגענט צו געבן געוויע און אינפארמירטע ענטפערס, ווייל ער באקומט צוטריט צו אָרגאַניזירטער דאטא. דאס איז באזונדערס נוצליך פאר קונה-סופאָרט, אינערליכע דאקומענטאציע און FAQ סיסטעמען.

דער קאנסאל פירט דעם באניצער דורך די שריט:

- אריינגעבן א נאמען און באשרייבונג פאר דער נאלעדזש-בעיס;
- אויפשטעלן צוטריט-דערלויבענישן;
- אויסקלייבן א דאטא-מקור, ווי Amazon S3, Salesforce, SharePoint אדער א Web Crawler.

דערנאך דארף די דאטא ווערן אינדעקסירט אין א וועקטאר-באזע, כדי מען זאל זי קענען שנעל און עפעקטיוו אריסנעמען. Amazon Bedrock שטיצט עטליכע וועקטאר-לאגער אפציעס: Amazon OpenSearch Serverless, Aurora PostgreSQL, MongoDB, Elastic און Pinecone. וועו א באניצער פרעגט דעם אגענט, נעמט די סיסטעם ארויס די שייכותדיגע אינפארמאציע פון דער נאלעדזש-בעיס און שטעלט זי אריין אינעם ענטפער.

סיכום פונעם Bedrock טייל (23:38:09)

דער Amazon Bedrock קאנסאל גיט א פולן אינטערפעיס צו פארוואלטן AI אגענטן, קאנפיגורירן וואָרקפלאוס און פארבינדן דאטא-מקורות. קאוד גיט פראגראמירער פולע קאנטראל איבער דער אויספירונג, אבער דער קאנסאל מאכט גרינגער דאס ארויסשטעלן, טעסטן און פיינע צופאסן, און מאכט עס צוגענגליך סיי פאר טעכנישע און סיי פאר נישט-טעכנישע באניצער. דורך צופאסן דעם מאדעל, גארדרעילס פאר זיכערקייט און נאלעדזש-בעיסעס פאר בעסערן פארשטאנד, זארגט Bedrock או AI לייזונגען בלייבן גענוי, עפעקטיוו און צוגעפאסט צו די געשעפט-באדערפענישן.

דערמיט ענדיגט זיך דער טייל "Introduction to Agentic Workflow". מען האט דורכגענומען די הויפט-באגריפן פון אגענטישע וואָרקפלאוס און ווי זיי פארבעסערן אויטאמאציע; מען האט געשאפן אן אגענט מיט Amazon Bedrock, פארבונדן מיט א CRM, וואס מאכט חשבונות און לייענט א FAQ מאנועל פאר ארויסנעמען אינפארמאציע; און מען איז דורכגעגאנגען דעם Bedrock קאנסאל פראקטיש.

אנהייבן מיט n8n: איינשרייבן און אויפשטעלן (23:39:11)

דערנאך גייט דער קורס איבער צו אויפשטעלן דעם ערשטן וואָרקפלאו אין n8n. מען עפנט דעם וועב-בראַוזער און גייט צו דער אפיציעלער n8n וועבזייטל. דארט קען מען זיך איינלאגירן אדער קליקן "Get Started". מען גיט אריין א פירמע-אימעיל, קליקט "Submit", און באקומט א באשטעטיגונג-אימעיל וואס מען דארף באשטעטיגן.

נאכן איינלאגירן באווייזט זיך די היים-זייט. ביים אנהייב איז עס אין דעם נארמאלן מאָדוס, אבער עס וועלן זיך אויפטאָן אַ פּאָפּ-אַפּ פֿענצטער וואו מען קען אויסקלויבן וואָס מען וויל צו בויען. אויב עס דערשיינט נישט, קען מען קליקן אויף "Workflows" אויבן לינקס. דאס איז דער ארט וואו אלע אייערע וואָרקפלאָוס וועלן זיך ווייזן.

דער ערשטער וואָרקפלאו: "Add first step" (23:40:00)

דער ערשטער שריט אין יעדן וואָרקפלאו איז צולייגן א "Trigger Node". דער קורס ערקלערט: דאס איז ווי דער פלאן פון דער וואָרקפלאו, דער שריט וואו דער וואָרקפלאו הייבט זיך אן. א טריגער איז א געשעעניש וואס לאזט אָן דעם וואָרקפלאו; א וואָרקפלאו לויפט נישט פון זיך אליין, סיידן מען לאזט אים אָן מיט דער האנט. עס איז ווי דער "לאנטש-קנעפל" אדער דער "אלארעם-גלעקל" פון דער וואָרקפלאו. וועלן מיר אויסקלויבן א טריגער וואָס וועט אנהייבן דעם וואָרקפלאו.

וואס איז א טריגער-נאוד (23:40:45)

דער ערשטער שריט איז שאפן א "Trigger Node". דער קורס ערקלערט: דאס איז ווי דער פלאן פון דער וואָרקפלאו, דער שריט וואו דער וואָרקפלאו הייבט זיך אן. א טריגער איז א געשעעניש וואס לאזט אָן דעם וואָרקפלאו; א וואָרקפלאו לויפט נישט פון זיך אליין, סיידן מען לאזט אים אָן מיט דער האנט. עס איז ווי דער "לאנטש-קנעפל" אדער דער "אלארעם-גלעקל" פון דער וואָרקפלאו: וועלן א געווייסע געשעעניש פאסירט אדער א באדינגונג ווערט דערפילט, קלינגט דער אלארעם און דער וואָרקפלאו הייבט אן ארבעטן.

די טריגער-סארטן וואס מען זעט:

- Trigger manually, מיט דער האנט;
- On app event, ביי א געשעעניש אין אן אפליקאציע;
- On a schedule: לויפט יעדן טאג, יעדע שעה אדער א באשטימטן אינטערוואל; וויכטיג פאר פעריאדישע וואָרקפלאוס;
- On webhook call;
- On form submission;
- On chat message, און נאך אנדערע.

אינעם ביישפיל קלייבט מען "On chat message", וואס לויפט דעם פלאו ווען דער באניצער שיקט א טשאט-מעסעדזש פאר די AI נאודס.

אויפשטעלן און טעסטן דעם טשאט-טריגער (23:42:15)

אויף דער לינקער זייט זעט מען Parameters און Settings. מען קען מאכן דעם טשאט עפנטליך צוגענגליך, און קאפירן דעם טשאט-URL פאר שפעטער. ביי Authentication זענען פאראן דריי סארטן: Basic Auth, User Auth און None. עס ווערט שטארק געראטן צו נוצן איינע פון די אויטענטיקאציעס, אבער אינעם ביישפיל נוצט מען None נאר צו ווייזן די הויפט-פונקציאנאליטעט.

מען קען אריינשרייבן אן "Initial Message " און עס צופאסן, למשל מיט א נאמען. דער מעסעדזש קען זיין אלס Expression אדער אין Fixed פארמאט; דא בלייבט עס Fixed. מען קען צולייגן פעלדער ווי: Allow File Uploads, Allowed File Types, Input Placeholder, Load Previous Session, Custom Chat Styling און נאך; דא לייגט מען צו Custom Chat Styling.

נאכן טעסטן זעט מען אז דער טריגער-נאוד איז געשאפן מיטן טעקסט "When chat message received". ווען מען פירט דעם מויז איבערן נאוד באווייזן זיך דריי אייקאנס: איינס צו לויפן דעם שריט, איינס צו דעאקטיווירן דעם נאוד, און איינס צו אויסמעקן. מען לויפט דעם נאוד, אויף דער לינקער זייט עפנט זיך א טשאט, מען שרייבט "hey", און דער נאוד לויפט מיט הצלחה; דער אויסגאנג אויפן עקראן איז "hey".

צולייגן אן AI Agent נאוד און א טשאט-מאדעל (23:44:16)

דער ווייטערדיגער שריט: מען קליקט אויפן פלוס, דערנאך "AI", און מען זעט א סך AI נאודס, ווי AI Agent, Anthropic, Google Gemini, Guardrails, OpenAI און נאך. מען קלייבט AI Agent. ביי "Source for Prompt" איז אויסגעקליבן "Connected Chat Trigger Node" (מען קען אויך קלייבן "Define below"). ווען מען קליקט "Execute step" קומט ארויס א פעלער: "A Chat Model sub-node must be connected and enabled".

דער קורס ערקלערט אז מען האט פארגעסן צולייגן דעם טשאט-מאדעל: ביי אן AI Agent לייגט מען צו ערשט א Chat Model, דערנאך Memory, און דערנאך Tool. מען הייבט אן מיטן טשאט-מאדעל. עס זענען פאראן א סך: Anthropic, Azure, AWS, DeepSeek, Google Gemini, Groq און נאך. מען קלייבט Google Gemini, ווייל מען קען אים נוצן אומזיסט מיט א פרייען API שליסל.

מען קליקט "Create new credential" און מען דארף אריינגעבן א Google Gemini API שליסל. צו באקומען אים גייט מען אין Google AI Studio, צום API Keys אפטייל אונטער דעם) Dashboardדארט זעט מען אויך שליסלען וואס זענען פריער געשאפן געווארן). מען קליקט "Create API key", גיט א נאמען צום שליסל, קלייבט א פראיעקט צו פארבינדן, און שאפט דעם שליסל. מען קאפירט אים, גייט צוריק צו n8n און פעיסט אים אריין.

ביי "Allowed HTTP Request Domains" קלייבט מען "All"; מען קען אויך קלייבן ספעציפישע דאמעינס אדער None, אבער "All" ווערט דא געראטן. מען קליקט Save, און עס באווייזט זיך אז די קרעדענשעלס זענען מיט הצלחה געשאפן.

א פראמפט מיט אן אויסדרוק און לויפן דעם נאוד (23:47:21)

צוריק אינעם AI Agent נאוד, קלייבט מען יעצט "Define below", און אנשטאט א פיקסירטן טעקסט נוצט מען אן Expression. דער פראמפט איז "What is the date today?", און מיט געווייזלטע קלאמערן לייגט מען צו א דאטום-צייט פארמאט, וואס באשטימט ווי דער רעזולטאט וועט אויסזען. מען לויפט דעם שריט, און דער נאוד לויפט מיט הצלחה. דערנאך גייט מען צוריק צום קאנוואס, וואו מען זעט ווי די פארבינדונג צווישן די נאודס זעט אויס.

n8n: זכרון, געצייג, פלענער און וואס איז אגענטישע AI (23:48:23)

דער אגענט אן זכרון: ער פארגעסט דעם פריערדיגן שמועס (23:48:23)

דער קורס ווייזט ווייטער ווי מען ארבעט מיט דעם AI-אגענט אין n8n. מען קען אים צופאסן לויט די אייגענע פארלאנגען. וועו מען לאזט לויפן דעם node, קומט ארויס אויפן עקראן אז די דאטום איז March 19, 2026, דאס הייסט די היינטיגע דאטום. וועו מען שיקט דעם מעסעדזש, ווערט דער node אויסגעפירט, און דער ענטפער איז: "לויט דעם timestamp וואס דו האסט געגעבן, איז די דאטום March 19, 2026."

דערנאך גייט מען צוריק צום AI agent און מען שטעלט צוריק די אפציע "connected chat trigger node". אזוי ארבעט דער אגענט מיט אלע פראמפטס וואס מען גיט אים, נישט נאר מיט א באשטימטן פראמפט וועגן דער דאטום. מען לאזט נאכאמאל לויפן, און דער node ווערט אויסגעפירט מיט הצלחה.

יעצט קומט א פראקטישער פרובירונג. מען שרייבט אין טשאט "the number is 10", און דער אגענט ענטפערט "okay, 10". דערנאך פרעגט מען "What is the number?", און מען זעט אז דער אגענט ווייסט נישט. ער האט פארגעסן די נומער 10 און דעם גאנצן פריערדיגן שמועס. ער קען נישט היטן דעם פריערדיגן זכרון. מען וויל אבער אז די פריערדיגע שמועסן זאלן בלייבן, כדי מען זאל קענען פירן א פארבונדענעם שמועס, אזוי ווי מען טוט מיט ChatGPT. דערפאר איז דא א באוונדערער באגריף: memory, זכרון.

צולייגן memory צום אגענט (23:49:55)

מען קליקט אויף דעם memory-טייל ביים אגענט. דארט זעט מען פארשידענע סארטן זכרון וואס AI-אגענטן שטיצן, למשל MongoDB Chat Memory, Postgres Chat Memory, Zep און נאך. פאר אנהייבער איז אבער שטארק רעקאמענדירט צו נוצן Simple Memory.

אין Simple Memory זענען דא עטלעכע פאראמעטערס:

- Session ID: מען לאזט עס אויף "connected chat trigger node". מען קען עס אויך שטעלן אויף "define below", אבער אין דער לעקציע בלייבט עס אויף connected chat trigger. דער session key קומט פונעם פריערדיגן node, און מען לאזט עס אזוי ווי עס איז.
- Context Window Length: עס שטייט אויף 5. דאס מיינט וויפיל פריערדיגע שמועסן דער מאדעל באקומט אלס קאנטעקסט, אין דעם פאל די לעצטע פינף שמועסן.

מען גייט צוריק צום canvas און מען זעט אז דער זכרון איז צוגעלייגט צום AI-אגענט. מען פרעגט נאכאמאל "What is the number?" און לאזט לויפן. דער workflow ווערט אויסגעפירט מיט הצלחה, און דאס מאל נעמט דער אגענט עס פונעם זכרון און ענטפערט "the number is 10". ער האט יעצט א זכרון וואס היט כאטש די לעצטע פינף שמועסן. אויף דעם אופן קען מען צולייגן א chat model, למשל Google Gemini אדער אנדערע, און א זכרון צום AI-אגענט.

צולייגן א געצייג (tool): Gmail Tool (23:51:28)

דער נעקסטער שריט איז צו לערנען ווי מען לייגט צו געצייג (tools). מען קליקט אויף Tool און מען זעט פארשידענע סארטן, למשל AI Agent Tool, Call n8n Workflow Tool, Code Tool, HTTP Request Tool, MCP Client Tool און נאך. אין דער לעקציע קלייבט מען א Gmail Tool, כדי ער זאל אליין צוגרייטן אן עמעיל. דער קורס באטאנט אז דאס איז א פאל פון ווירקלעכן לעבן: ווי מען קען אויטאמאטיש צוגרייטן א Gmail-בריוו מיט א n8n workflow.

די שריט:

- מען זוכט "Gmail Tool" און מען קליקט אויף אים. עס עפנט זיך דער פענצטער פון דעם געצייג.
- מען דארף שטעלן די credentials. מען קליקט אויף דעם credential -טייל. ביים פרעזענטירטן חשבון איז Gmail שוין פארבונדן, דערפאר שטייט אז דער חשבון איז connected.
- ביי א נייעם באניצער דארף מען קליקן "Sign in with Google" און קלייבן איינעם פון די חשבונות. מען ווערט געבעטן דעם login ID און פאסווארט, און דערנאך פארבינדט זיך n8n מיטן Google-חשבון אויטאמאטיש. מען קליקט continue.
- ביי "Allowed HTTP Request Domains" איז שטארק רעקאמענדירט צו קלייבן All, און דערנאך save.

שטעלן דעם געצייג און דעם פראמפט (23:53:33)

דערנאך דארף מען געבן א tool description. מען קען עס שרייבן מיט דער האנט, אבער אין דער לעקציע שטעלט מען עס אויטאמאטיש, לויט דעם resource און דער operation. ביים resource קלייבט מען דעם טייל פון צוגרייטן א בריוו (draft), און ביי operation קלייבט מען "Get Message". דארט איז אויך דער Draft ID. מען קען צולייגן נאך אפציעס, למשל Attachment Prefix און Download Attachments, אבער אין דער לעקציע איגנארירט מען זיי.

דערנאך גייט מען צוריק צום AI agent. ביי "Source for Prompt (User Message)" קלייבט מען "Define below", דאס הייסט א צוגעפאסטער פראמפט, און מען שרייבט דעם user message: "hey we are offering discounts". מען לאזט לויפן דעם node, און ער ווערט אויסגעפירט מיט הצלחה.

מען גייט צוריק צום canvas און מען זעט אז אן עמעיל-node איז פארבונדן צום AI-אגענט, מיטן נאמען "Get a message in Gmail". דער Gmail-חשבון איז שוין פארבונדן מיטן n8n-דעמא. אין "Define below" גיט מען נאך א מעסעדזש: "hey there, your workflow name is offering discounts". דאס וועט אנקומען אלס draft-בריוו אין Gmail. אויף דעם אופן קען מען אויטאמאטיש צוגרייטן בריוו, אדער שיקן עמעיל צו אזויפיל מענטשן ווי מען וויל.

מען קען אויך צולייגן נאך אפציעס: System Message, Max Iterations, Return Intermediate Steps, Enable Streaming, Batch Processing און נאך. אין דער לעקציע לאזט מען זיי איבער. דער node ווערט אויסגעפירט מיט הצלחה.

קאנטראלירן אין Gmail און סיכום פונעם workflow (23:55:05)

כדי זיך צו פארזיכערן גייט מען אריין אין דעם Gmail -חשבון וואס איז פארבונדן מיטן פראיעקט, אין דער Drafts-אפטיילונג. מען זעט אז א בריוו איז צוגעגרייט געווארן מיטן טעקסט "hey there my workflow 3 is offering discounts".

אזוי פארבינדט מען פארשידענע nodes מיטן AI-אגענט. אלע דריי nodes זענען אויסגעפירט געווארן מיט הצלחה: צוערשט דער chat model, דערנאך דער זכרון, און דערנאך א געצייג. אלס געצייג האט מען גענוצט א געוועלכן ביישפיל, Gmail. דער קורס דערמאנט אז מען קען אויך נוצן א weather API, דאס הייסט פארבינדן אן אנליין וועטער-געצייג און זען ווי ער ארבעט. פאר אנהייבער קען מען אויך ארבעטן מיט form responses. דער Gmail-ביישפיל איז נאר געוויזן געווארן ווי א ביישפיל ווי מען פארבינדט אן עמעיל-געצייג מיטן אגענט, און עס האט געארבעט אן א פעלער.

דער קורס רעקאמענדירט שטארק צו אויספארשן די פארשידענע מעגלעכקייטן פון n8n, ווייל עס איז זייער אינטערעסאנט. מען זאל נישט נאר זיצן און איבערחזרן דאס וואס איז געוויזן געווארן. מען זאל פרובירן צולייגן מער nodes און מער געצייג, און זען ווי זיי ארבעטן איינער מיטן אנדערן.

די פלענער פון n8n (23:56:35)

וועז די פרייַע פרובע-צייט ענדיגט זיך, קען מען אויפגרעידן. מען קליקט אויף upgrade, און עס זענען דא דריי פלענער:

- Starter: בערך 24 א חודש. מען באקומט 2.5 טויזנט workflow executions , אומבאגרעניצטע אקטיוועע workflows, איין shared project, פינף concurrent executions , אומבאגרעניצטע באניצער, איין טאג workflow history, און נאך.
- Pro: בערך 60 אייראס א חודש. מען באקומט בערך 10 טויזנט workflow executions, אומבאגרעניצטע אקטיוועע workflows, דריי shared projects, 20 concurrent executions, 7 טעג insights, admin roles, global variables, און נאך.
- Enterprise: פאסיג פאר געשעפטן.

מען זאל קלייבן לויט די אייגענע באדערפענישן, און איבונגען אזוי פיל ווי מעגלעך אין די 14 טעג פון דער פרובע. לויטן קורס, וועז דער טייל איז קלאר, ווערט אלץ אנדערש פיל גרינגער צו פארשטיין.

וואס איז אגענטישע AI? (23:58:09)

עס הייבט זיך אן א נייער טייל. איינע פון די גרעסטע ענדערונגען אין טעכנאלאגיע היינט איז אז ווייכווארג ווערט פון בלויז ענטפערן אויף פראגעס צו טאקע העלפן מענטשן פארענדיגן ארבעט. אויפן פשוטסטן אופן: אגענטישע AI איז א סיסטעם וואס גיט נישט נאר א ענטפער, נאר פרובירט ווייטער צו שטופן א ארבעט. אנשטאט נאר זאגן "דא איז דער ענטפער", קען זי פארשטיין דעם ציל, קוקן אויפן מצב, באשליסן אויף א שריט, און אים אויספירן.

א פשוטער ביישפיל פונעם טאג-טעגליכן לעבן: א געוועלכער אסיסטענט קען דיר זאגן ווי דער ביליגסטער פליען איז. א סיסטעם אין אגענט-סטיל וואלט פארגלייכן די פליען, קוקן אויפן בודזשעט, צופאסן צו די באליבטע צייטן, ווארענען אויב די באגאזש-כללים זענען שלעכט, און דערנאך העלפן געפינען די פרטים פאר דער באשטעלונג. דאס איז דער אונטערשייד: עס פירט זיך ווייניגער ווי א זוך-קעסטל און מער ווי א געהילף וואס פרובירט פארענדיגן עפעס צוואמען מיט דיר.

אין Python וועלט דאס אינטערעסאנט, ווייל Python מאכט עס גרינג צו פארבינדן לאגיק, געצייג, אקציעס אין בלעטערער, פיילס און באשלוסן, אלץ אויף איין פלאץ. דערפאר נוצן אזויפיל מענטשן Python צו בויען קלוגע אסיסטענטן, בלעטערער-געהילפן, פארשונג-געצייג און workflow-באטס. דער הויפט-געדאנק צו געדענקען: אגענטישע AI באדייט אקציע לויט א ציל. דאס איז ווייכווארג וואס קען באטראכטן, באשליסן און טון עפעס נוצליכס שריט נאך שריט, אנשטאט נאר געבן טעקסט.

טראדיציאנעלע AI קעגן אגענטישע AI (23:59:40)

כדי דער אונטערשייד זאל זיין פראקטיש און נישט נאר טעאָרעטיש, פארגלייכט דער קורס מיט דער עלטערער סארט AI. טראדיציאנעלע AI איז געוועלך שמאל: זי איז געבויט צו טון איין ארבעט גוט. זי קען קלאסיפיצירן

בילדער, רעקאמענדירן ווידעאס, פאראויסזאגן פארלאנג, אדער ענטפערן א פראגע. זי גיט א רעזולטאט און מען נוצט אים. אבער אין רוב פעלער דארף א מענטש אדער אן אנדער סיסטעם באשליסן וואס צו טון ווייטער.

אגענטישע AI איז אנדערש, ווייל זי פארבינדט עטלעכע שריט צוזאמען. זי קוקט אויפן ציל, קאנטראלירט דעם יעצטיגן מצב, און באשליסט וועגן דעם נעקסטן שריט. אויב נויטיג נוצט זי א געצייג, קאנטראלירט דעם רעזולטאט, און גייט ווייטער. דער עיקר-אונטערשייד איז נישט אז זי איז ווי א כישוף קליגער, נאר אז זי איז מער אויסגעריכט אויף אקציע.

צוויי משלים:

- טראדיציאנעלע AI איז ווי א קאלקולאטאר: מען פרעגט און עס ענטפערט. אגענטישע AI איז ווי א יונגער מיטארבעטער: מען גיט אים א ריכטונג, און ער הייבט אן ארבעטן אויף דער ארבעט שריט נאך שריט.
- טראדיציאנעלע AI איז ווי א מאפע וואס ווייזט א וועג. אגענטישע AI איז ווי א שאפער וואס באמערקט דעם טראפיק, טוישט דעם וועג, פילט אן ברענשטאף וועז נויטיג, און פרובירט דיך ברענגען צום ציל אין צייט.

פארוואס דאס איז אזוי וויכטיג אין דער וויקלעכקייט (24:00:41)

דער טעם איז פשוט: מענטשן ווילן נישט נאר ווייכווארג וואס רעדט שיין, זיי ווילן ווייכווארג וואס שפארט צייט. אויב א סיסטעם קען פארמינערן איבערגעחזרטע ארבעט, באהאנדלען רוטינע נאכפאלגונגען, זאמלען אינפארמאציע און האלטן די ארבעטן אין גאנג, האט עס גלייך א ווערט.

ביישפילן:

- קונה-שטיצע: א געוועלכער סיסטעם קען ענטפערן אויף איין פראגע. א סיסטעם אין אגענט-סטיל קען לייענען די פראבלעם, קאנטראלירן די פרטים פון דער באשטעלונג, ארויסנעמען אינפארמאציע פון די כללים, צוגרייטן דעם ענטפער, ווייטערגעבן צו א מענטש נאר וועז עס איז נויטיג, און האלטן דעם פאל דערהיינטיגט.
- שאפן אינהאלט: א שאפער קען נוצן א קלוגן סיסטעם צו זאמלען פארשונג, ארגאניזירן פונקטן, אנליין צוגרייטן א סקריפט, צוגרייטן אידייען פאר א thumbnail, און אויך שאפן א טשעקליסט פאר ארויפלייגן.

דערפאר איז די טעמע אזוי פאפולער: זי איז גלייך פארבונדן מיט דעם וואס מענטשן פרובירן טון יעדן טאג. מען שפארט צייט, מען פארגרעסערט די פראדוקציע, און מען פארמינערט רוטינע מי.

א צווייטער טעם איז צוטריטלעכקייט. אמאל האט אויטאמאציע געפאדערט שטייפע געצייג, און אסאך פון זיי האבן געפאדערט אינזשעניר-ארבעט. היינט קענען מענטשן שנעלער בויען נוצליכע סיסטעמען פאר ארבעט, בפרט מיט Python, ווייל Python לאזט פארבינדן פיילס, וועבזייטלעך, APIs, ספרעדשיטס און אייגענע לאגיק אן צופיל שווערigקייטן. דער גרויסער לימוד דא: אגענטישע AI איז וויכטיג ווייל זי פארוואנדלט ווייכווארג פון פאסיווע ענטפער-מאשינען אין אן אקטיוון ארבעטס-געהילף, און דאס איז א זייער גרויסע ענדערונג. דערנאך גייט דער קורס איבער צום יסוד וואס מאכט יעדן אגענט ארבעטן, און הייבט אן צעלייגן דעם סיסטעם אויף זיינע טיילן.

אגענטן: בוי-שטיינער, ריזיקעס און דער ערשטער דעמא (24:02:11)

דריי בוי-שטיינער: אגענט, סביבה און אינטעראקציע (24:02:11)

דער קורס צעטיילט דעם גאנצן ענין אויף דריי פשוטע שטיקלעך, און דא ווערט די טעמע אסאך גרינגער צו לערנען.

- דער אגענט: דאס איז דער "טוער". עס איז דער טייל פון סיסטעם וואס באקומט דעם ציל, פרובירט צו מאכן פארשריט, און מיט פשוטע ווערטער איז ער דער וואס נעמט די באשלוסן.
- די סביבה (environment): דאס איז אלעס וואס ארום דעם אגענט, וואס ער קען זען אדער באאיינפלוסן. דאס קען זיין א וועבזייטל, א טעקע, אן אינבאקס, א ספרעדשיט, א קאסטומער-טיקעט אדער אפילו א שפיל-סקרין.
- אינטעראקציע: דאס איז דאס הין-און-צוריק צווישן דעם אגענט און דער סביבה. דער אגענט זעט עפעס, טוט א שריט, באקומט דעם רעזולטאט, און דערנאך באשליסט ער וואס צו טון ווייטער.

עס ווערט ערקלערט מיט א ביישפיל פון עסן-באשטעלן. דער אגענט איז דער געהילף וואס פרובירט אריינצולייגן די באשטעלונג. די סביבה איז די עסן-אפליקאציע: דער מעניו, די אדרעס וואו מען זאל ברענגען, און דער באצאל-בלאט. די אינטעראקציע איז דער גאנצער פראצעס: קוקן אויף די אפציעס, אויסקלייבן זאכן, אנוועגדן פילטערס, אריינלייגן די באשטעלונג, און באשטעטיגן דעם רעזולטאט. דער כלל איז אז עס איז שטענדיג פאראן א טוער, א וועלט אין וועלכער ער ארבעט, און א רינג (loop) וואס פארבינדט ביידע.

ווי אגענטן פלאנירן, האנדלען און לערנען (24:03:11)

דער קורס רופט דאס "דאס הארץ-קלאפן פון גאנצן סיסטעם". נאכדעם וואס מען קען די דריי הויפט-שטיקער, ווערט ערקלערט ווי דער גאנצער פראצעס גייט פאר מיט דער צייט. אן אגענט ארבעט געוועגליך אין א רינג: ערשט קוקט ער אויף דעם מצב, דערנאך באשליסט ער וואס צו טון, דערנאך טוט ער דעם שריט, און דערנאך קוקט ער אויף דעם רעזולטאט. נאכדעם גייט ער אדער ווייטער, אדער ער טוישט דעם כיוון, אדער ער שטעלט זיך אפ. דאס איז דער גרונט-פלוס: באמערקן, טראכטן, האנדלען און איבערקוקן.

א פראקטישער ביישפיל: די אויפגאבע איז "געפין מיר א לעפטאפ וואס פאסט אריין אין מיין בודזשעט, מיט א גוטער באטעריע-לעבן". דער סיסטעם קען זוכן אפציעס, פארגלייכן די פרייזן, אוועקנעמען די שוואכע ברירות, צוזאמשטעלן א קורצע איבערזיכט פון די בעסטע, און פרעגן נאכפאלג-שאלות ווען עפעס איז נישט קלאר. דאס איז פלאנירן און האנדלען.

וואו קומט אריין דאס לערנען? עס ווערט באטאנט אז "לערנען" מיינט נישט שטענדיג שווערע טעאריע. אין אסאך פראקטישע סיסטעמען מיינט לערנען פשוט זיך פארבעסערן לויט רעאקציעס (feedback). למשל, אויב דער באניצער זאגט "שפילן אינטערעסירט מיך נישט, איך וויל נאר באטעריע און וואג", פאסט דער סיסטעם צו זיין קומענדיגן באשלוס. אויב ער פאלט דורך אויף איין וועבזייטל, קען ער פרובירן א צווייטן וועג. און אויב דער רעזולטאט איז געווען שוואך, קען ער פון דאסניי פלאנירן דעם קומענדיגן פרואוו.

פאפולערע מוסטערן ווי צו בויען אגענטן אין Python (24:04:12)

נאכדעם ווערט געוויזן עטליכע געוועגליכע אופנים ווי מענטשן שטעלן אויף אגענטן אין Python. דאס איז וויכטיג ווייל א גוטע סטרוקטור איז אפט וויכטיגער ווי א שיינער אינפאל.

• פלאנירער און ארבעטער (planner and worker): איין טייל באשליסט די שריט, און דער צווייטער טייל פירט זיי אויס. דאס איז נוצלעך ווען די אויפגאבע איז גרויס און מען דארף זי קלאר צעטיילן.

• האנדלען דורך געצייג (tool based action): דא טוט דער סיסטעם אליין נישט אלעס דירעקט. אנשטאט דעם קלייבט ער אויס פון א סעט געצייג. איין געצייג קען זוכן אויפן וועב, איינס קען לייענען א טעקע, איינס קען צוזאמפאסן טעקסט, און נאך איינס קען שיקן אן אפדעיט.

• זכרון פלוס האנדלען (memory plus action): אין דעם אויפשטעל האלט דער סיסטעם נוצלעכע נאטיצן פון פריערדיגע שריט. דאס העלפט אויסמיידן איבערגעחזרטע טעותים און העלפט ביי די וווייטערדיגע באשלוסן.

די מוסטערן זענען פאפולער ווייל זיי האלטן Python אגענט-פראיעקטן ריין און בייגעוודיג, און מען קען זיי גרינגער פארריכטן (debug).

ריזיקעס און פאראנטוואַרטליכער באנוץ (24:05:12)

דער קורס זאגט אז מען וועט איצט זיין ערליך וועגן דעם וואס קען גיין שלעכט. דער טייל איז וויכטיג, ווייל וואס מער האנדלונגען א סיסטעם קען טון, אלץ מער פארזיכטיג דארף מען זיין.

• ערשטע ריזיקע, א פאלשע האנדלונג: אויב דער סיסטעם פארשטייט נישט ריכטיג די אויפגאבע, קען ער פונדעסטוועגן טון עפעס מיט זיכערקייט און שאפן אמת'ע פראבלעמען. א שלעכטע איבערזיכט איז איין זאך. שיקן א פאלשע מעסעדזש צו א קונה איז אסאך ערגער.

• צווייטע ריזיקע, צוטריט (access): אויב א סיסטעם קען עפענען געצייג, טעקעס, וועבזייטלעך אדער אקאונטס, זענען די ערלויבענישן זייער וויכטיג. מען טאר אים נישט לאזן אנרירן אלעס נאר ווייל ער קען.

• דריטע ריזיקע, פריוואטקייט: אויב א סיסטעם לייענט אימעילס, דאקומענטן אדער דאטא פון קונים, דארפן מענטשן וויסן וואס ער קען זען, וואס ער האלט אפ, און וואוהין די אינפארמאציע גייט.

פאראנטוואַרטליכער באנוץ מיינט שטעלן קלארע גרענעצן: געבן דעם סיסטעם נאר דעם צוטריט וואס ער דארף באמת, האלטן לאגס (רשימות פון וואס ער האט געטון), פרעגן א מענטשן אויף באשטעטיגונג ביי וויכטיגע האנדלונגען, און אויספרובירן מיט זיכערע ביישפילן פאר מען לאזט עס ארויס אויף אמת. דער כלל: וואס מער פרייהייט מען גיט אן אגענט, אלץ מער שוץ-גרענעצן (guardrails) דארף מען ארום אים.

פראקטישע שווערigקייטן ביי זעלבסטשטענדיגע סיסטעמען (24:06:14)

דערנאך ווערט באהאנדלט די פראקטישע שווערigקייטן וואס מענטשן טרעפן ווען זיי בויען סיסטעמען וואס האנדלען אליין ביי לענגערע אויפגאבעס.

• פארלירן דעם ציל: דער סיסטעם הייבט אן גוט, אבער נאך עטליכע שריט קען ער זיך פארדרייען צו זייטיגע אויפגאבעס אדער פארברענגען צייט אויף האנדלונגען פון ווייניג ווערט.

• אומאָרדנטליכע דאטא פון דער אמת'ער וועלט: וועבזייטלעך טוישן זיך, קנעפלעך רוקן זיך אוועק, און טעקע-פארמאטן זענען פארשידן.

• וויסן ווען אויפצוהערן: געוויסע סיסטעמען פרובירן צו לאנג, אנדערע הערן אויף צו פרי. אויסצופלאנירן די ריכטיגע באלאנס איז שווערער ווי עס זעט אויס.

• צוטרוי: אסאך מענטשן געפינען די סיסטעמען אינטערעסאנט, אבער זיי זארגן נאך אלץ וועגן געוויקייט, אויפזיכט און קאנטראל.

• קאסטן און שנעלקייט: מער שריט, מער רופן פון געצייג, מער קאנטראלן און מער זכרון קענען מאכן דעם סיסטעם לאנגזאמער אדער טייערער.

דערפאר, ווערט באטאנט, איז א דורכגעטראכטער פלאן אזוי וויכטיג.

די צוקונפט פון אגענטן (24:07:16)

דער קורס זאגט אז די קומענדיגע כוואַליע איז נישט נאר גרעסערע שמועס-געצייג, נאר מער פארבונדענע געהילפן וואס קענען ארבעטן אריבער פארשידענע אפליקאציעס, געדענקען נוצלעכן קאנטעקסט, און שטיצן אמת'ע ארבעטס-פלוסן במשך פון צייט.

• מערשריטיגע דיגיטאלע געהילפן: אנשטאט איין ענטפער וועלן מענטשן ערווארטן אז סיסטעמען זאלן אדורכפירן א סדר, ווי פארשן, פארגלייכן, צוזאמפאסן, צוגרייטן און ווייטער.

• אגענטן וואס ארבעטן צוזאמען: איין סיסטעם קען זאמלען אינפארמאציע, א צווייטער קען זי איבערקוקן, און א דריטער קען צוגרייטן דעם לעצטן רעזולטאט. דאס איז באזונדערס נוצלעך ביים שרייבן קאוד, אין אפעראציעס, קונה-סערוויס, און ארבעט וואס פארלאנגט אסאך פארשונג.

• שטארקערע זיכערהייט-שיכטן: ווי די סיסטעמען טוען מער, לייגן מענטשן אלץ מער אכט אויף ערלויבענישן, אויפזיכט און באשטעטיגונג-סיסטעמען.

• מען וועט מסתמא אויך זען מער פערזענליכע ארבעטס-געהילפן פאר שאפער, פראגראמירער, גרינדער פון פירמעס, סטודענטן און קליינע געשעפטן. דערפאר, ווערט געזאגט, איז די טעמע אזוי פאפולער אנליין: מענטשן זענען גלייך ווי עס קען זיי שפארן מי אין זייער טעגליכער ארבעט.

דער קורס ענדיגט דעם טייל אז די צוקונפט איז נישט פארבייטן מענטשן מיט איין ריזיגע מאשין. די מער רעאליסטישע צוקונפט איז מענטשן וואס ארבעטן צוזאמען מיט דיגיטאלע געהילפן וואס נעמען איבער די איבערגעחזרטע שריט, בשעת די מענטשן בלייבן אין קאנטראל איבער דעם כיוון און דעם אורטייל.

בעסטע מעטאדן ווי צו בויען גוט פון ערשטן טאג (24:08:20)

• ערשטנס, הייבט אן קליין: הייבט נישט אן מיט א סיסטעם וואס פרובירט טון צען זאכן. הייבט אן מיט איין ענגע, נוצלעכע אויפגאבע, למשל צוזאמפאסן סופָארט-טיקעטס, קאנטראלירן אפדעיטס אין טעקעס, אדער זאמלען פראדוקט-פרטים פון עטליכע בלעטער.

• צווייטנס, באשטימט קלאר וואס הייסט הצלחה: ווי וועט איר וויסן אז דער סיסטעם האט געארבעט? האט ער געשפארט צייט? האט ער פארענדיגט די אויפגאבע? האט ער אויסגעמיטן טעותים? אויב הצלחה איז אומקלאר, וועט די פארבעסערונג אויך זיין אומקלאר.

• דריטנס, האלט מענטשן אין רינג ביי וויכטיגע האנדלונגען: לאזט דעם סיסטעם פארשלאגן, שרייבן א טיוטה און צוגרייטן, אבער פארלאנגט אן איבערקוק פאר שיקן, אויסמעקן, קויפן, אדער טוישן סענסיטיוו דאטא.

• פערטנס, פארשרייבט אלעס וויכטיגע אין לאג: אויב א סיסטעם האט געזוכט, געקליקט, אויסגעקליבן, אדער געשריבן עפעס פאר אייך, דארפט איר שפעטער קענען נאכגיין דעם וועג. דאס איז נויטיג סיי פארן פארריכטן און סיי פאר צוטרוי.

• פינפטנס, פלאנירט פאר דורכפאל: נעמט אן אז לינקס קענען זיך צעברעכן, געצייג קענען זיך אפשטעלן פון צו לאנג ווארטן (time out), אדער די באניצער-בקשות קענען זיין נישט פולשטענדיג. לייגט צו נאכאמאל-פרואוון (retries), רעזערוו-שריט, און קלארע באדינגונגען ווען אפצושטעלן.

- זעקסטנס, מאכט די אנווייזונגען פשוט: וואס קלארער דער ציל, אלץ בעסער דער רעזולטאט. אפילו דער בעסטער סיסטעם האט שווערigקייטן וועלן די אויפגאבע אליין איז זייער אומקלאר.

דער סיום פון דעם טיל: שטארקע אגענטן זענען געווענליך נישט די מערסטע קאמפליצירטע, נאר די וואס האבן א קלארן ציל, ריינע גרענעצן, און א סטאבילע אויספירונג.

וואו ווייטער צו לערנען, און א טשעלענדזש (24:09:51)

די פראגע איז, וואוהין דארפן מענטשן גיין נאך דער לעקציע? דער בעסטער לערן-וועג איז א מישונג פון דריי זאכן: ערשטנס, שטארקע יסודות אין Python; צווייטנס, פראקטישע קליינע פראיעקטן; דריטנס, אקטיווע קהילות וואו מענטשן טיילן אמת'ע ביישפילן און פראקטישע טעותים. א פשוטער בראוזער-געהילף, א טעקע-סדרן, א פארשונג-געהילף, אדער א סארטירער פון סופֿאָרט-טיקעטס קענען לערנען מער ווי פאסיוו צוקוקן.

פאר קהילות קען מען נאכפאלגן Python קהילות, פראגראמירער-פארומס, GitHub פראיעקט-בלעטער, און YouTube קאנאלן וואס בויען אמת'ע דעמאס שריט ביי שריט. עס איז אויך כדאי צו נאכפאלגן די אפיציעלע בלאגס און דאקומענטאציע פון די גרויסע פלאטפארמעס, ווייל די געביטן טוישן זיך שנעל און נייע ארבעטס-פלוסן ווייזן זיך כסדר.

צום סוף שטעלט דער קורס א טשעלענדזש: לאזט נישט די טעמע בלייבן א געדאנק. בויט איין קליינעם ארבעטנדיגן אגענט אין Python. נישט קיין ריזיגע פלאטפארמע, נישט קיין פערפעקטער סטארטאפ-איינפאל, נאר איין נוצלעכער פראיעקט. ביישפילן: א סיסטעם וואס לייענט א פאלדער און סארטירט טעקעס לויט זייער סארט; א פשוטער געהילף וואס פארגלייכט פראדוקטן פון א קליינער רשימה און גיט א רעקאמענדאציע; אדער אן אויפגאבע-געהילף וואס לייענט א בקשה, צעטיילט זי אויף שריט, און פאלגט נאך דעם פארשריט. דער ציל איז נישט פערפעקציע, נאר דורכצולעבן דעם פולן רינג: געבן דעם ציל, לאזן דעם סיסטעם האנדלען, איבערקוקן דעם רעזולטאט און אים פארבעסערן, און אים אויך מאכן זיכערער.

אויפשטעלן דעם ארבעטס-פלאץ פארן דעמא (24:11:24)

דער דעמא וועלט געבויט אויף א זייער פראקטישן אופן. אנשטאט גלייך אריינשפרינגען אין א גרויסן פראיעקט, שטעלט מען ערשט אויף א ריינעם Python ארבעטס-פלאץ אין VS Code, מען אינסטאלירט נאר די ביבליאטעקן וואס מען דארף באמת, און מען פרובירט אויס יעדן טייל באזונדער.

די ביבליאטעקן זענען אויסגעקליבן געווארן צוליב סטאביליטעט און קלארקייט:

- Google רעקאמענדירט דעם Google GenAI SDK אלס דעם אפיציעלן Python SDK פאר Gemini.
- דער איצטיגער Google פעקל פון LangChain הייסט langchain-google-genai.
- Gymnasium איז דער אנגעהאלטענער פארבייט פארן אלטן gym פעקל.
- pyautogui אינסטאלירט זיך דירעקט פון PyPI אויף Windows.
- פאר קאנטראל איבערן בראוזער ווערט גענוצט Selenium, ווייל דער מאדערנער Selenium נוצט Selenium Manager צו באהאנדלען די דרייווערס, און ער איז מער פארלעסלעך פאר לעבעדיגע דעמאס ווי אויטאמאציע נאר דורכן קלאוויאטור.
- עס ווערט גענוצט Gemini 2.5 Flash אנשטאט די עלטערע 2.0 ווערסיעס, ווייל די דאקומענטאציע פון Google ווייזט 2.5 Flash אלס די סטאבילע אפציע.

עס ווערט געזאגט אז מען שאפט דעם פראיעקט-פאלדער און מען אינסטאלירט אלעס פונעם VS Code טערמינאל אויף Windows. אין דער לעקציע ווערן אבער געוויזט פולגנדע באפעלן:

- מען שאפט א פאלדער מיט mkdir אונטערן נאמען python_agent_demo , און מען גייט איין אים מיט cd.
- מען אינסטאלירט Python מיט sudo apt install python.
- מען שאפט א ווירטועלע סביבה (virtual environment), און מען אקטיווירט זי מיט: source venv/bin/activate
- מען לאדנט אראפ די ביבליאטעקן מיט pip install, אנגעהויבן מיט google-genai.

נאכדעם וואס אלע פעקלעך זענען אינסטאלירט, איז די סביבה גרייט. פונדעסטוועגן ווערט געווארנט אז מען זאל נישט אננעמען אז אלעס ארבעט; מען דארף שטענדיג קאנטראלירן אז דער אויפשטעל איז ריכטיג און אז אלע פעקלעך זענען ריכטיג אינסטאלירט.

דער API שליסל אויף א זיכערן אופן (24:13:26)

דערנאך לייגט מען צו דעם API שליסל אויפן זיכערסטן אופן: אין הויפט-פאלדער (root) פונעם פראיעקט שאפט מען א טעקע .env. און מען לייגט דארט איין דעם שליסל. דעם API שליסל קען מען באקומען פון Google אליין; מען זאל זוכן דעם פרייען API שליסל וואס Google גיט. נאכן איינלייגן דעם שליסל גייט מען ארויס פון דער טעקע. עס ווערט געווארנט אז די טעקע דארף בלייבן לאקאל אויף דער מאשין, זי טאר קיינמאל נישט איינגיין אין git , און מען טאר זי נישט טיילן אין סקרינשאטס. דערנאך שרייבט מען א פשוטע Python טעקע וואס קאנטראלירט צוויי זאכן; די ערשטע פון זיי איז צי מען קען לייענען דעם שליסל.

פון סעטאפ-קאנטראל ביז וועב-אגענט (24:13:56)

קאנטראלירן דעם סעטאפ און דער ערשטער רוף צום מאדעל (24:13:56)

אין דעם טייל פונעם קורס ווערט קודם געוויזן ווי אזוי מען פרעגט נאך צוויי זאכן: ערשטנס, צי דער סביבה (environment) איז ריכטיג אויפגעשטעלט, און צווייטנס, צי מען קען מאכן א געלונגענעם רוף צום מאדעל.

פאר דעם ווערן געוויזן צוויי טעקעס. די ערשטע איז requirements.txt, וואו עס שטייט די ליסטע פון אלע פעקלעך (packages) וואס זענען אינסטאלירט. ווער עס וויל וויסן וועלכע פעקלעך מען דארף, קען דארט נאכקוקן. די צווייטע איז די .gitignore טעקע, וואו מען האט איינגעלייגט די נויטיגע אינהאלטן.

דערנאך ווערט באשאפן א טעקע מיטן נאמען setup_check.py. זי איז געמאכט כדי צו קאנטראלירן צי דער סעטאפ איז ריכטיג. דער קאוד הייבט אן מיט import os און ברענגט איין אלע נויטיגע מאדולן. דערנאך גייט מען צוריק צום טערמינאל און מען לאזט עס לויפן.

ווייל עס איז גוט געלאפן, זעט מען א גאר קורצן ענטפער פונעם מאדעל, אז דער סביבה איז גרייט. אין אסאך פעלער זעט מען אויך אונטן די צאל טאָקענס (token counts).

דער קורס ערקלערט דעם קאוד מיט פשוטע ווערטער:

- די פונקציע load_dotenv לייענט די באהאלטענע ווערטן פון דער .env טעקע.
- os.getenv ברענגט דעם שליסל (key) איין אין Python.

- די פונקציע genai.Client שאפט א פארבינדונג-אביעקט.
- client.models.generate_content שיקט איין קליינע בקשה צום מאדעל.
- צום סוף דרוקט response.text דעם ענטפער אויפן טערמינאל.

דער ערשטער אמת'ער אגענט: FrozenLake און די Q-טאבעלע (24:15:00)

נאכדעם וואס מען האט באשטעטיגט אז דער סביבה און דער שליסל ארבעטן, גייט מען איבער פון סעטאפ צו דער ערשטער אמת'ער אגענט-דעמא. דערפאר ווערט באשאפן א טעקע first_agent.py. די פראגראם באטראכט א מצב, קלייבט אויס א שריט (action), און ווערט בעסער מיט יעדן נייעם פרואוו.

מען הייבט אן בכוונה מיט א קליינעם "גריד" (א נעץ פון קעסטלעך). דער טעם: עס איז זייער וויזועל און גרינג צו ערקלערן, און עס לאזט זיך קאנצענטרירן אויפן החלטה-קרייז (decision loop). דער קאוד הייבט זיך אן מיט import random און from typing, און ענדיגט זיך מיט main. מען לאזט לויפן די טעקע מיט דעם געהעריגן באפעל און מען זעט דעם רעזולטאט.

דער קורס ערקלערט דעם קאוד פשוט:

- FrozenLake-v1 איז א קליינע גריד-וועלט.
- די פראגראם הייבט אן מיט א Q-טאבעלע פול מיט נולן. מען קען זיך פארשטעלן די Q-טאבעלע ווי א פונקטן-בלאט: יעדע שורה איז א מצב (state), יעדע זייל איז א שריט (action), און יעדע צעל האלט ווי נוצליך יענער שריט איז געווען אין יענעם מצב.
- די פונקציע choose_action באשליסט צי דער אגענט זאל פרובירן עפעס נייעס, אדער ניצן די בעסטע בריה וואס ער האט שוין געלערנט. דאס קאנטראלירט epsilon: א הויכער epsilon מיינט מער אויספארשן (exploration), א נידעריגער epsilon מיינט אז דער אגענט פארלאזט זיך אויף וואס ער האט שוין געלערנט.
- אין דער פונקציע train_agent שטעלט דער אגענט צוריק דעם סביבה (reset), קלייבט שריט, באקומט באלוינונגען (rewards), און דערהיינטיגט די Q-טאבעלע נאך יעדן שריט.
- די שורה וואס דערהיינטיגט די טאבעלע איז דאס הארץ פונעם לערן-פראצעס: זי נעמט דעם אלטן פונקט, פארגלייכט אים מיטן נייעם רעזולטאט, און רוקט דעם פונקט אין דער ריכטונג פון דעם וואס האט בעסער געארבעט.
- נאכן טריינירן לויפט evaluate_agent איין טעסט-עפיזאד מיט די בעסטע שריט פון דער טאבעלע. אזוי קען מען ווייזן דעם עולם דעם וועג וואס דער אגענט האט געלערנט.

יסודות פון reinforcement learning (24:17:01)

דערנאך ווערט די אידעע געמאכט מער באדייטנספול, דורך קלאר ערקלערן דעם לערן-שריט דערהינטער. אין דעם פונקט פון דער דעמא שטעלט מען זיך אפ און מען ערקלערט אז reinforcement learning איז פשוט: פרואוו, פידבעק און אויסגלייכן. דער אגענט לייענט נישט קיין לערן-בוך; ער לערנט פון איבערגעחזרטע פרואוון.

עס זענען דא בלויז עטליכע ווערטן וואס מען דארף געדענקען:

- alpha קאנטראלירט ווי שנעל דער אגענט דערהיינטיגט וואס ער גלייבט.

- gamma זאגט ווי וויכטיג די צוקונפטיגע באלוינונגען זענען.
- epsilon קאנטראלירט די צופעליגקייט, וואס העלפט דעם אגענט אנטדעקן בעסערע וועגן פון אנהייב אן.

די וויכטיגסטע שורה איז דער דערהיינטיגונג-כלל (update rule). מען האט שוין געהאט אן אלטע שאצונג ווי נוצליך א שריט איז. דערנאך האט מען געטון דעם שריט, געזען די באלוינונג, און געקוקט אויף דער בעסטער צוקונפטיגער ברירה פונעם קומענדיגן מצב. דאס אלעס פאראייניגט מען אין א בעסערער שאצונג.

די באציאונג מיטן סביבה פאסירט אין דער פונקציע evaluate_agent, און דאס איז דער טייל וואס מען האלט זיכטבאר אויפן עקראן בשעת מען ערקלערט דעם רעזולטאט. מען זעט אין רעזולטאט די ריכטונגען ,left down, right און up. דער סביבה וועלט צוריקגעשטעלט, דער אגענט קוקט אויפן יעצטיגן מצב און קלייבט דעם בעסטן באקאנטן שריט פון דער Q-טאבעלע. דער סביבה ענטפערט מיטן קומענדיגן מצב און מיט דער באלוינונג, און די פראגראם דרוקט וואס עס איז פאסירט, כדי מען זאל קענען נאכגיין דעם גאנצן וועג.

אריינברענגען LLMs פאר טראכטן און קאנטעקסט (24:18:02)

יעצט גייט מען איבער צום קומענדיגן שטאפל, וואו א פראגראם קען אדורכטראכטן א בקשה פון א באניצער און אויסקלייבן א נוצליכן קומענדיגן שריט, דורך אריינברענגען LLMs ווי GPT פאר טראכטן (reasoning) און קאנטעקסט. ביז אהער האט דער אגענט געלערנט בלויז פון באלוינונגען אין א קליינעם גריד. יעצט וועלט וועט געוויזן ווי מען לייגט צו טעקסט-באזירטע טראכטונג אויבן אויף דעם. אנשטאט בעטן די פראגראם צו גיין לינקס אדער רעכטס, בעט מען זי צו באטראכטן אן אמת'ע בקשה פון א באניצער און צו באשליסן וועלכע סארט שריט זאל קומען דערנאך.

פאר דעם טייל ניצט מען LangChain מיט Gemini. דער ציל איז פשוט: געבן דעם מאדעל א קלארע אויפגאבע, געבן אים דעם ריכטיגן קאנטעקסט, און בעטן אים אויסצוקלייבן איין שריט פון א גאר קליינער ליסטע פון ערלויבטע שריט.

די טעקע הייסט reasoning_agent.py. מען הייבט אן מיט import json, import os און נאך, און מען ענדיגט דעם קאוד. דערנאך לויפט מען אים אינעם טערמינאל מיט python און דעם נאמען פון דער טעקע. עס קומט ארויס א טעות אז langchain-google-genai איז נישט אינסטאלירט. מען אינסטאלירט עס, לאזט ווידער לויפן, און דער קאוד ארבעט.

דער קורס זאגט אז דער ביישפיל איז גוט ווייל ער איז גרינג צו ערקלערן: דער מאדעל טוט נישט אלעס אליין. ער קלייבט בלויז אויס פון א קליינער ליסטע פון זיכערע שריט וואס מען האט שוין פריער דעפינירט אין Python. דאס האלט די דעמא פארשטענדליך און פארמינערט איבעראשונגען.

- דער ChatPromptTemplate בויט א סטרוקטורירטן פראמפט מיט פיר טיילן: די ערלויבטע געצייג (tools), דער ציל פונעם באניצער, די פריערדיגע היסטאריע, און דער יעצטיגער מצב.
- דער מאדעל גיט צוריק א קליינעם JSON-אביעקט וואס זאגט וועלכן שריט צו טון און פארוואס.
- דער Python-קאוד קאנטראלירט צי דער צוריקגעגעבענער שריט איז גילטיג, און לויפט די פאסיגע פונקציע.

דער מאדעל העלפט אלזא מיטן טראכטן, אבער די פראגראם האלט נאך אלץ די קאנטראל איבער דער אויספירונג.

א קלארער פראמפט און א ריינער שריט-פלוס (24:19:33)

דער קומענדיגער טייל מאכט דעם פראמפט קלארער און דעם שריט-פלוס ריינער. דער קורס באטאנט אז גוטע רעזולטאטן קומען נישט פון א לאנגן, שיינעם פראמפט. זיי קומען געווענליך פון א פראמפט וואס איז קלאר, שמאל און גרינג צו באשטעטיגן. אין דעם הויפט-פראמפט-בלאק פון דער טעקע שטייט, למשל, אז מען זאל אויסקלייבן דעם איינציגן בעסטן קומענדיגן שריט פון א געוויסער ליסטע.

פארוואס ארבעט עס גוט פאר א לעבעדיגער דעמא?

- ערשטנס, דער system message דעפינירט די פונקטליכע ארבעט.
- צווייטנס, ער באגרעניצט דעם ענטפער-פארמאט צו JSON.
- דריטנס, דער human message טיילט אפ דעם ציל, די היסטאריע און דעם יעצטיגן מצב.

דאס מאכט די אויפגאבע גרינגער פארן מאדעל נאכצוגיין, און גרינגער פאר אונז צו געפינען טעותים (debug). דערנאך שאפט מען די chain אין איין ריינער שורה. דאס איז בלויז א ציכטיגער אופן צו זאגן: ערשט בוי דעם פראמפט, און דערנאך גיב אים איבער צום מאדעל. וועז דער רעזולטאט קומט צוריק, צעלייגט מען אים (parse), באשטעטיגט אים, און גייט ווייטער.

דער פלוס איז יעצט גאר קלאר: מען נעמט דעם קאנטעקסט פונעם באניצער, שאפט א קלארן סטרוקטורירטן פראמפט, באקומט איין שריט, און לויפט דאס Python-געצייג.

זכרון, געצייג און א זיכערער "fallback" (24:20:35)

כדי דער אסיסטענט זאל זיך פילן מער פולשטענדיג, לייגט מען צו זכרון (memory), געצייג, און א פשוטן קרייז פון אויסקלייבן שריט. דער קורס וויל אז דער עולם זאל פארשטיין אז זכרון דארף נישט קיין קאמפליצירטע געצייג כדי צו זיין נוצליך. אין דעם טייל פון דער דעמא האלט מען בלויז א קורצע ליסטע פון פריערדיגע שריט און פון דעם וואס די געצייג האבן צוריקגעגעבן.

אין קאוד זעט מען צוויי memory.append פונקציעס. יעדעס מאל וואס די פראגראם טוט א שריט, געדענקט זי וואס זי האט פונקט געטון. שפעטער, וועז דער מאדעל דארף אויסקלייבן דעם קומענדיגן שריט, ווערט דער זכרון אריינגעלייגט אינעם פראמפט אלס טייל פון דער היסטאריע.

די געצייג זענען בכוונה גאר פשוט. יעדעס געצייג איז א געווענליכע Python-פונקציע מיט א קלארער ארבעט:

- איינע זוכט נאך אין נאטיצן (notes).
- איינע שאצט אפ ווי דרינגענד א זאך איז (urgency).
- איינע שרייבט א ענטוווורף פון א ענטפער.

מער איז נישטא. דאס אויסקלייבן פון שריט ווערט קאנטראלירט אין א באזונדערן טייל פונעם קאוד. דער fallback איז זייער וויכטיג: אויב דער מאדעל גיט צוריק עפעס אומגעריכט, פאלט די פראגראם נישט אוועק (crash). זי קלייבט פשוט אויס א זיכערן דיפאלט-שריט. דאס איז די סארט פארזיכטיגע פראגראמירונג (defensive coding) וואס איז וויכטיג ביי א לעבעדיגער דעמא.

אן אויטאמאטישער AI-אסיסטענט פאר וועב-אויפגאבעס (24:21:38)

דער לעצטער טייל פארבינדט אן אויפגאבע אין בלעטערער (browser), כדי צו בויען אן אויטאמאטישן AI-אסיסטענט פאר וועב-אויפגאבעס. אנשטאט צו אויטאמאטיזירן א צופעליגע עפנטליכע וועבזייטל וואס קען זיך

מארגן טוישן, אויטאמאטיזירט מען א לאקאלע HTML-בלאט אלס ציל. אזוי בלייבט די דעמא סטאביל, און דער עולם קען דאך זען דעם גאנצן פלוס פון בלעטערער-אויטאמאציע.

ערשט שאפט מען א פשוטן לאקאלן בלאט, demo_page.html, און מען שרייבט אריין דעם נויטיגן קאוד. דערנאך שאפט מען דאס Python-אויטאמאציע-סקריפט און לאזט עס לויפן. וואס דאס סקריפט טוט איז גאר קלאר:

- ערשט בעט עס דעם מאדעל צו פארוואנדלען א פשוטע בקשה אויף ענגליש אין א קליינעם JSON-אביעקט.
- דערנאך עפנט Selenium דעם לאקאלן בלאט.
- Python פילט אויס יעדעס פעלד איינס נאכן אנדערן, און דריקט אויפן submit-קנעפל.
- עס לייענט דעם רעזולטאט אויפן בלאט און דרוקט אים צוריק אויפן טערמינאל.

דאס גיט א פולן קרייז: פארשטיין די בקשה, זי סטרוקטורירן, טון אין בלעטערער, און אויפקלייבן דעם זיכטבארן רעזולטאט.

אויפגאבעס מיט עטליכע שריט און פידבעק (24:22:40)

כדי די דעמא זאל זיין מער רעאליסטיש, ווייזט מען ווי אזוי מען באהאנדלט אויפגאבעס וואס פאדערן מער ווי איין שריט און קענען אריינרעכענען פידבעק. די פשוטע ערקלערונג: די ערשטע שורה איז די אריגינעלע אויפגאבע, און די צווייטע שורה איז א פארבעסערונג אדער אן עקסטרע אנווייזונג. וועז ביידע גייען אריין אינעם פראמפט, גיט דער מאדעל צוריק א פולשטענדיגן JSON-אביעקט, און די פארעם אין בלעטערער ווערט אויסגעפילט מיט בעסערע פרטים.

דער קורס באטאנט א וויכטיגן פונקט: דער מאדעל דריקט גארנישט אליין. ער גרייט בלויז צו רייניע, סטרוקטורירטע דאטא. די אמת'ע ארבעט אין בלעטערער ווערט נאך אלץ געטון דורך Python און Selenium. די דאזיגע אפטיילונג איז וואס מאכט דעם פלוס גרינג צו געטרויען און גרינג צו פארריכטן.

אכטונג געבן און פארבעסערן מיט דער צייט (24:23:11)

דער לעצטער שריט ווייזט ווי אזוי מען היט נאך וואס עס איז פאסירט, און ווי מען פארבעסערט דאס פירונג מיט דער צייט. ווי פשוט א פראיעקט זאל נאר זיין, דארף מען האבן א רעקארד וואס עס איז פאסירט בשעת דעם לויף. דערפאר האט מען צוגעלייגט א קליינע לאג-פונקציע:

def log_message(...)

די פונקציע שרייבט די וויכטיגע שריט אריין אין logs/web_agent_log.txt, און דרוקט זיי אויך אויפן טערמינאל. ביי א לעבעדיגער דעמא איז דאס נוצליך, ווייל עס ווייזט פונקטליך וואס די פראגראם טוט במשך דער גאנצער אויפגאבע: וועז דער בלאט האט זיך געעפנט, וואס דער בלעטערער האט צוריקגעגעבן, און וועז דער לויף האט זיך געענדיגט.

דאס פארבעסערן קומט פון דריי קליינע אידעען. די ערשטע: האלטן דעם אויסגאנג פונעם פראמפט סטרוקטורירט, כדי דאס צעלייגן (parsing) זאל זיין גרינגער. דער טראנסקריפט ווערט אבער אפגעשניטן אינמיטן דער צווייטער אידעע, און די ווייטערדיגע פרטים קומען נישט פאר אין דעם אפשניט.

בריט דאטא און Xpander AI: דאטא און אגענטן (24:24:12)

סוף פון דעם בראוזער-דעמא און א קורצער איבערבליק (24:24:12)

דער קורס ענדיקט דעם פריערדיקן דעמא מיט עטלעכע עצות. צווייטנס, האלט די ערלויבטע אקציעס שמאל, כדי דער מאדעל זאל נישט קענען אוועקוואנדערן צו ווייט. דריטנס, לייג שטענדיג צו א זיכערן "fallback", א רעזערוו-וועג, פאר דעם פאל אז דער אויסגאנג וואס קומט צוריק איז נישט וואס מען האט געריכט.

צום שליסן דעם דעמא ווערט איבערגעחזרט וואס מען האט געטון:

- מען האט צוגעגרייט א ריינע Python סביבה אין VS Code.
- מען האט געבויט א קליינעם לערנענדיקן אגענט.
- מען האט צוגעלייגט טעקסט-באזירטע ריזאנירונג מיט א קאנטראלירטן סעט פון אקציעס.
- צום סוף האט מען גענוצט די ריזאנירונג צו פירן א בראוזער-ארבעט אויף א לאקאלן בלאט.

אזוי האט מען באקומען א פולן ביישפיל פון אנהייב ביז סוף. דערנאך הייבט זיך אן א נייע טעמע: איידער מען בויט עפעס, דארף מען פארשטיין פון וואו די דאטא קומט.

וואס איז Bright Data (24:24:42)

עס ווערט ערקלערט אז מען קען זיך פארשטעלן Bright Data ווי דעם מערסט פארלעסליכן מאטאר פאר וועב-דאטא אויף דער וועלט. מען זעט דעם אינטערפייס נאכן איינלאגירן. דארט קען מען שרייבן סיי וועלכן פראמפט, און עס ווערן געוויזן פארגעלייגטע פראמפטן. אויף דער לינקער זייט איז דא א נאוויגאציע-שטאנג מיט "data sets", "scrapers" און "web access". אז מען קליקט אויף "data sets" און דערנאך אויף "data set marketplace", באקומט מען א סך פארשידענע קאטעגאריעס פאר דער דאטא וואס מען וויל.

דער קורס באטאנט אז Bright Data איז נישט נאר א "scraping" געצייג, א געצייג וואס ציט אויס דאטא פון וועבזייטלעך. עס איז א גאנצע פלאטפארמע, געבויט ארום דריי הויפט-זאכן:

- ערשטנס, א ריזיגע פראקסי-נעץ. אונטער "my proxies" זעט מען "dashboard", "event log" און "playground". אין דעם דעמא האט מען נאך נישט געשאפן קיין פראקסי. כדי צו זען דעם דעשבאָרד און דעם "event log" מוז מען ערשט שאפן א פראקסי. די פראקסיס זענען "residential", "mobile" און "data center". זיי לאזן צוקומען צו יעדן וועבזייטל אן ווערן בלאקירט אדער דערקענט.
- צווייטנס, א Web Scraper API. אונטער "scrapers" איז דא א "scrapers library", וואו מען קען דורכקוקן פארשידענע scrapers. ווייטער איז דא "AI gateway" און "web access", וואו מען קען צוקומען צו סיי וועלכן web access API. דאס זענען גרייטע "endpoints", אנקומען-פונקטן, צו ציען סטרוקטורירטע דאטא פון פלאטפארמעס ווי Amazon, LinkedIn, Instagram און הונדערטער אנדערע.
- דריטנס, דער "data set marketplace" וואס מען האט פריער געזען. דאס איז א ביבליאטעק פון גרייטע דאטא-סעטן צום אראפלאדן, אין פארשידענע געשעפט-געביטן: e-commerce, פינאנצן, marketplace, real estate (גרונט-אייגנטום), סאשעל מידיא, רייזעס, פרייע דאטא-סעטן און נאך. די דאטא ליגט שוין גרייט און וואַרט.

וואס מאכט Bright Data באזונדער, לויט דעם קורס, זענען צוויי זאכן: גרייס און "compliance", דאס הייסט האלטן זיך צו די געזעצן. מען באקומט א גראנטיע פון 99.9% "uptime", אז דער סיסטעם ארבעט שטענדיג.

די פלאטפארמע שטימט אויך פולשטענדיג מיט GDPR און CCPA. דאס מיינט אז מען באקומט די דאטא נישט נאר שנעל, נאר אויך אויף א פאראנטווארטליכן אופן, מיט פולער לעגאלער שטיצע.

וואס איז Xpander AI (24:26:46)

דער קורס ערקלערט אז דאטא אליין איז נישט גענוג. מען דארף עפעס וואס זאל זי באמת פראצעסירן, פארשטיין און ארויסגעבן "insights", דאס הייסט נוצליכע פארשטאנדן פון איר. דא קומט אריין Xpander AI. מען זעט די היים-בלאט, וואו מען קען קליקן אויף "free trial" און אנהייבן.

Xpander AI איז א "back-end as a service" פלאטפארמע, ספעציעל געבויט פאר AI אגענטן. אויף דער בלאט זעט מען אויסדרוקן ווי "pre-built AI workflows", "custom AI workflows", א AI טראנספארמאציע וואס רירט אן יעדע באוונדערע פירמע, "personal AI agents" און "specialized AI agents".

דער קורס ערקלערט עס פשוט. געווענליך, אז מען וויל בויען אן אגענט וואס ציט אויס דאטא, פראצעסירט זי און גיט ארויס insights, דארף מען שרייבן קאוד, פירן די סערווערס, אויפשטעלן א פלאן וועז עס זאל לויפן, באהאנדלען פעלערן, און אליין אויפהאלטן דעם גאנצן סיסטעם. דאס איז וואכן לאנג אינזשעניר-ארבעט. Xpander נעמט דאס אלעס אוועק:

- "Agent Studio", וואו מען קען בויען, טעסטן און ארויסשטעלן AI אגענטן אויף א וויזועלן אופן.
- א וויזועלער אגענט-בויער, וואו מען קען צייכענען גאנצע "workflows" אן שרייבן איין שורה קאוד.
- API אינטעגראציע, כדי צו פארבינדן געצייג ווי Bright Data מיט עטלעכע קליקס.
- א "secret manager", כדי צו האלטן די API שליסלען זיכער אויף איין ארט.
- "scheduling", "logging" און "versioning". דאס זענען פלאנירונג, פארשרייבונג און ווערסיעס, און אלעס ווערט באהאנדלט אויטאמאטיש אין הינטערגרונט.
- "multi-agent" שטיצע, וואס מיינט אז עטלעכע אגענטן קענען ארבעטן צוזאמען אויף דער זעלבער "pipeline".

עס ווערט געזאגט אז סיי א פראגראמירער וואס וויל פולע קאנטראל, סיי א נישט-טעכנישער מארקעטינג-מענטש וואס וויל נאר רעזולטאטן, פאר ביידע איז Xpander געבויט. אז מען פארבינדט עס מיט דער קוואליטעט פון Bright Data's דאטא, האט מען א באמת שטארקע קאמבינאציע.

אויסקלייבן און אראפלאדן דעם Instagram דאטא-סעט (24:28:16)

איצט הייבט זיך אן דער לעבעדיקער דעמא. מען גייט אין Bright Data צו "data sets" און דערנאך צו "data set marketplace". מען זוכט אין די קאטעגאריעס און קלייבט אויס "social media". דער ציל איז צו געבן דעם אגענט א פראמפט וואס זאל ארויסנעמען insights פון א גרויסן סך דאטא, צו שאפן קעפלעך ("captions") פאר פאוסטס, אדער צו געפינען וויכטיגע געשעפט-insights פון די דאטא-סעטן.

מען קלייבט אויס דעם "Instagram posts" דאטא-סעט. מיט אים קען מען דורכזוכן פרטים פון פאוסטס, אריינגערעכנט ווי שטארק די באניצער באטייליגן זיך, hashtags, קאמענטארן און מעדיע-אינהאלט. פאפולערע נוצן זענען אנאליז ווי גוט אינהאלט ארבעט, "sentiment analysis" (אנאליז פון געפילן) און "audience targeting" (צילן אויף א געוויסן עולם) אויף Instagram.

די פרטים פון דעם דאטא-סעט:

- בערך 42 דאטא-פעלדער.
- סך-הכל רעקארדס: 272.98 מיליאן.
- פרישע רעקארדס זענען צו באקומען דורך דעם web scraper.
- פרייז: בערך 0.0025 דאלער פער רעקארד.
- מינימום באשטעלונג: 250 דאלער.

מען קויפט עס נישט אין דעם מאמענט, מען דארף נאר די דאטא צום אראפלאדן. מען קען זען אלע רעקארדס און אנשטעלן פילטערס, לויט ווי מען ווייל, למשל לויט URL. מען קען צולייגן א כלל ("rule") און אויסקלייבן פעלדער: hashtags, נאר באשרייבונגען, likes, בילדער. דא דארף מען נישט קיין פילטער, ווייל מען ציט אלע דאטא.

מען קליקט אויף "download sample". מען קען אפהיטן ווי CSV אדער ווי JSON טעקע. עס ווערט רעקאמענדירט צו אראפלאדן ווי CSV, האט מען געקליקט אויף "download as CSV". נאך א ווייל ווערט די מוסטער-דאטא געהעריג אראפגעלאדן.

שאפן דעם אגענט אין Xpander AI (24:30:20)

מען גייט צוריק צו Xpander AI און קליקט אויף "free trial". מען זעט דעם הויפט-דעשבאָרד מיט "custom agents", אן אנזאג אז די פרייע פראבע-צייט ענדיקט זיך אין 14 טעג, און עטלעכע אגענטן וואס זענען שוין פריער געשאפן געווארן. מען קען אויך שאפן א "personal agent" דורך "new agent".

די טריט:

- מען קליקט אויף "new folder" און רופט די טעקע "Instagram post bot", דערנאך "create folder".
- מען עפנט די טעקע און זעט אז עס זענען נאך נישטא קיין אגענטן.
- מען קליקט אויף "new agent", גיט דעם נאמען "IG support agent bot", און קליקט "create new agent".

נאך א ווייל לאדט זיך דער אינטערפייס, מיט דעם אגענט און זיין "configuration".

די קאנפיגוראציע: LLM, ריזאנירונג, צוטריט און אויסגאנג (24:31:22)

אין "LLM settings" זעט מען אז דער גרויסער שפראך-מאדעל איז אריגינעל Anthropic. מען קען אויך אויסקלייבן OpenAI, Google AI Studio, Microsoft Foundry, Amazon Bedrock, Fireworks און נאך. אין דעם דעמא קלייבט מען אויס OpenAI, מיטן מאדעל GPT 5.5. מען קען נעמען אנדערע מאדעלן, אבער דער לעצטער איז GPT 5.5, און דעם האט מען אויסגעקליבן.

- "reasoning effort": מען קען נעמען "low", אבער מען נעמט "medium" פאר בעסערע ריזאנירונג.
- "access": "all users in my account" מיינט אז יעדער וואס ניצט דעם חשבון האט צוטריט צום אגענט. "only me" איז פאר מער פריוואטקייט. מען קלייבט אויס "all users", כדי די אנדערע וואס באקומען צוטריט זאלן קענען קאנטראלירן דעם אגענט.

- "output format": ס'איז דא "default", "markdown" און "structured output". מיט "markdown" ענטפערט דער אגענט מיט טעקסט אויסגעשטעלט אין markdown. מיט "default" ענטפערט ער מיט פרייען טעקסט. מען קלייבט אויס markdown.

שוץ-געלענדער, מולטי-אגענט און וויסן-באזעס (24:32:22)

- "LLM guardrails", שוץ-געלענדער: מען קען אנצינדן "PII detection", אנטדעקן פערזענליכע אינפארמאציע און זי פארדעקן. וויטער איז דא "prompt injection detection", אנטדעקן פרואוון אריינשמוגלען שעדליכע אנווייזונגען, און "OpenAI moderation".
- "multi-agent": מען קען צובינדן אנדערע אגענטן פון דעם ארבעטס-ארט ("workspace") צו מאכן א מולטי-אגענט אָרקעסטראציע.
- "knowledge bases": עס איז דא Xpander's אייגענע "built-in vector database" און "add knowledge base".
- וויטער זענען דא "templates", "notifications" און א "danger zone", וואס מען דארף נישט.

איידער מען גייט צו "instructions", היט מען אפ די ענדערונגען פון דער אלגעמיינער קאנפיגוראציע.

דערנאך גייט מען צו "knowledge bases" און קליקט "create new". מען גיט דעם נאמען "Instagram post", און א באשרייבונג "post details, captions". די באשרייבונג איז אפציאנאל. מען קליקט "create", דערנאך "upload file", און מען שלעפט אריין ("drag and drop") די CSV טעקע וואס מען האט אראפגעלאדן פון Bright Data. מען קליקט "upload". עס ווערט געמאלדן "files uploaded successfully", און מען זעט ווי די CSV טעקע ווערט פראצעסירט. דאס איז די וויסן-באזע.

געצייג, קאנאלן, זכרון און די אנווייזונגען (24:34:54)

מען גייט צוריק צו "agents", עפנט "IG support agent", און קוקט די וויטערדיקע אפציעס:

- "tools": אריינגעבויטע געצייג וואס מען קען צולייגן. נאך ענדערונגען דארף מען "deploy" כדי צו דערהיינטיגן. ביישפילן: שיקן אימעיל, וועב-זוך, ארויפלאדן טעקעס, שאפן בילד פון טעקסט, "sleep", "local DB", און ארויסנעמען טעקסט.
- "channels": מען קען טעסטן דעם API. ס'איז אויך דא SDK, chat, Slack, webhook, MCP און A2A.
- "memory": מען צינדט אן "session storage". אונטער "user memories" לאזט מען דעם אגענט האלטן זכרון פאר יעדן באניצער. אונטער "agent memories" לאזט מען אים האלטן זיין אייגענעם זכרון איבער אלע שמועסן. ס'איז דא "max tool calls from history", און "session summary", וואס פארקירצט אויטאמאטיש לאנגע שמועסן אין א קורצן סך-הכל פאר שפעטער. דאס איז אפציאנאל.

אין "instructions" באשרייבט מען די ראלע און פערזענלעכקייט פון דעם אגענט. דאס באשטימט דעם טאן פון זיינע ענטפערס. דער פראמפט אין דעם דעמא זאגט: דו ביסט א סאשעל-מידיא דאטא אנאליטיקער. אנאליזיר דעם ארויפגעלאדענעם Instagram דאטא-סעט און גיב מיר א סטרוקטורירטן באריכט מיט:

- איבערבליק: וויפיל פאוסטס מען האט אנאליזירט, דורכשניטליכע likes פער פאוסט, דורכשניטליכע קאמענטארן.
- די שפיץ 5 פאוסטס: קעפל, likes, קאמענטארן, "shares", דאטום, hashtags.

- די שפיץ 10 hashtags: נאמען, וויפיל מאל יעדער איז ארויסגעקומען, און דורכשניטליכע likes אויף פאוסטס וואס ניצן אים.

אגענט-באריכט און ChatGPT-ס זכרון (24:36:25)

דער סוף פון דעם Instagram-אנאליז אגענט (24:36:25)

דער קורס ענדיגט דא אויפצושטעלן דעם אגענט וואס אנאליזירט Instagram-דאטא. מען שרייבט אינעם אויפגאבע-טעקסט אז דער אגענט זאל ארויסציען די העכסטע פאסטן, די hashtags און business insights, און אז ער זאל ארויסגעבן דעם רעזולטאט אין ריינע אפטיילונגען מיט קעפלעך (headers).

עס ווערט ערקלערט אז מען קען אויך צולייגן עטליכע "goals": וואס דער אגענט דארף אויפטון, אנווייזונגען, שריט-ביי-שריט אנפירונג, כללים, אדער באגרעניצונגען (constraints). יעדע אנווייזונג ווערט צוגעלייגט צום system prompt.

דערנאך קומט דער פעלד "expected output", וואו מען באשרייבט וויאזוי דער ענטפער פונעם אגענט זאל אויסזען. דאס ווערט צוגעלייגט צום סוף פונעם system message, כדי צו באשטימען א גאר געארדנטן פארמאט. דער קורס באמערקט אז דאס איז נישט קיין מוז, ווייל דער אגענט ענטפערט סיי-ווי אויף א געארדנטן אופן. אבער וועט מען וויל באקומען א געווישן סארט רעזולטאט, קען מען דאס אנווייזן גלייך דארט.

מען דריקט "save", און דער אגענט ווערט אפגעהיטן. דערנאך מאכט מען צו דעם זייטן-באלקן (sidebar).

דער אגענט ארבעט אויף דער CSV-טעקע (24:36:55)

מען גייט אריין אינעם chat-טייל, וואו עס שטייט "How can I help you today?". מען הענגט צו די ערשטע CSV-טעקע וואס מען האט פריער אראפגעלאדן, אז דער אגענט זאל בעסער פארשטיין די דאטא. דערנאך שרייבט מען דעם פראמפט:

- פון דער אויסגעשאבטער (scraped) דאטא, ציה ארויס די 5 העכסטע פאסטן לויט engagement;
- די 10 מערסט-באניצטע hashtags;
- דריי business insights פון דער דאטא.

דער אגענט דארף ערשט באארבעטן דעם דאטא-סעט, און דאס נעמט א ווייל. נאכדעם דריקט מען enter און מען ווארט עטליכע מינוט, בשעת דער LLM "טראכט". נאך עטליכע מינוט ווייזט ער אז ער האט געטראכט און גיט ארויס דעם רעזולטאט.

מען זעט די שריטן וואס דער אגענט האט דורכגעמאכט:

- ערשט האט ער אונטערגעזוכט די סטרוקטור פונעם דאטא-סעט;
- דערנאך האט ער דורכגעפירט א פולע אנאליז, און אנאליזירט די hashtags און די קוואליטעט פון דער דאטא;
- ער האט דורכגעקוקט אלע רעקארדס און ארויסגענומען דעם גאנצן דאטא-סעט;
- צום סוף האט ער צוזאמגעשטעלט דעם ענדגילטיגן באריכט.

דער רעזולטאט: א פולער Instagram-באריכט (24:37:56)

דער באריכט הייבט אן מיט אן איבערבליק (overview): וויפיל פאסטן מ'האט אנאליזירט, א דורכשניט פון ארום 29,000 likes, א דורכשניט פון 3,226 קאמענטארן פער פאסט, און א דורכשניטלעכער engagement rate פון ארום 757.63%. דער אגענט האט אויך צוגעלייגט א קורצע באמערקונג, אז דער דאטא-סעט נעמט ארום א גאר ברייטן פארנעם: פון קליינע שאפער (micro creators) מיט וויניגער ווי 500 נאכפאלגער, ביז ריזיגע בראנדס מיט 22 מיליאן און מער.

דערנאך קומען די 5 העכסטע פאסטן, אויסגעשטעלט לויט likes. ביי יעדן פאסט ווערט אנגעוויזן: likes, סארט פאסט, נאמען פונעם קאנטע, קאמענטארן, shares און דאטום (פאסט 1, פאסט 2, פאסט 3 א.א.וו.).

דערנאך קומען די 10 העכסטע hashtags, אין א טאבעלע מיט: ראנג, hashtag, וויפיל מאל באניצט, דורכשניטלעכע likes, און קאטעגאריע.

נאכדעם קומען די business insights: insight 1, insight 2 און insight 3. צום סוף האט דער אגענט אויך געגעבן א פולן באריכט וואס מען קען אראפלאדן.

אין דער לעקציע ווערט געזאגט אז מען האט נישט געריכט אזויפיל פון דעם אגענט, און אז ער האט קלאר איבערגעשטיגן די ערווארטונגען. דער פארמאט איז זייער געארדנט און פול מיט דאטא-פונקטן, און דער לינק צום אראפלאדן איז זייער נוצליך. מען קען דעם באריכט אויך פארעפנטליכן גלייך פון דארט: מען דריקט "edit", מען קוקט איבער די ענדערונגען, און מען פארעפנטליכט. דער קורס באטאנט אז אזא סארט אנאליז וואלט א דאטא-אנאליטיקער געמוטמען 2 ביז 3 טעג אויסצוארבעטן מיט די הענט, בשעת דער אגענט האט עס געטון אין ווייניגער ווי עטליכע מינוט.

וואס איז דער זכרון (memory) אין ChatGPT (24:39:30)

דא הייבט זיך אן א נייער טייל. דער זכרון אין ChatGPT לאזט דעם AI געדענקען אינפארמאציע וואס דער באניצער האט שוין געגעבן אין פריערדיגע שמועסן. דאס פארבעסערט וויאזוי ChatGPT רעדט מיט באניצער, ווייל ער באניצט זיך מיט פריער-דערמאנטע פרטים כדי צו געבן מער צוגעפאסטע און שייכותדיגע ענטפערס. עס ווערט געגעבן א משל: עס איז ווי א דיגיטאלער געהילף וואס געדענקט דיינע פארליבטקייטן און פריערדיגע שמועסן, כדי יעדער שמועס זאל זיין גלאטער און מער פערזענליך.

וויאזוי מען צינדט אן דעם זכרון:

- מען דריקט אויף דעם באניצער-נאמען;
- מען גייט צו settings;
- מען דריקט אויף personalization;
- דארט געפינט מען דעם memory-פעלד און מען צינדט עס אן;
- מיט "manage" קען מען קוקן צי ער האט שוין אפגעהיטן עפעס ווענן דיר.

מען קען אויך אויסמעקן דעם זכרון פון ChatGPT. עס ליגט אינגאנצן אין די הענט פונעם באניצער: צי צו האלטן דעם זכרון, אים אויסמעקן, אדער אויסקלייבן וועלכע זכרונות צו האלטן און וועלכע אויסצומעקן.

עס ווערט דערמאנט אז די פונקציע איז צוגענגליך פאר "Plus"-באניצער און enterprise-באניצער. אין דער באשרייבונג שטייט אז ChatGPT וועט ווערן מער בעהילפיג וואס מער מען שמועסט, ווייל ער כאפט אויף פרטים און פארליבטקייטן כדי צוצופאסן זיינע ענטפערס. מיט "learn more" קומט מען צו דער

אפיציעלער וועבזייטל פון OpenAI, וואו עס ווערט באשריבן "memory and new controls" פאר ChatGPT.

דוגמאות פון OpenAI און ווי עס ארבעט (24:41:03)

אויפן וועבזייטל זעט מען דאס זעלבע "manage memory" פעלד. ביים באניצער אינעם קורס איז דארט נאך גארנישט אפגעהיטן, אבער אין דער דוגמא פון OpenAI זענען דא זכרונות ווי "האט א 2-יעריגע טאכטער מיטן נאמען Lena" און "די טאכטער Lena האט ליב jellyfish". אזעלכע זאכן ווערן אפגעהיטן. עס ווערט ערקלערט אז מען קען ChatGPT בעטן ער זאל געדענקען עפעס ספעציפיש, אדער לאזן אז ער זאל אליין אויפכאפן פרטים, און די פרטים ווערן באניצט אין שפעטערדיגע שמועסן.

דער קורס זאגט צו צו ווייזן פראמפטס וויאזוי דער זכרון קען העלפן אין פרייאריטיזירן ארבעט, אין א פיטנעס-פלאן, אדער ביי א פראיעקט.

אונטער "You're in control" ווערט געוויזן וויאזוי מען באניצט דעם זכרון און מעקט אים אויס, און וויאזוי מען קוקט אים איבער אין "manage". עס איז אויך דא א "temporary chat": ווען מען דריקט אויף ChatGPT 4 ארויף אינעם מעניו, איז דא אן אפציע "temporary chat". ווען מען צינדט עס אן, שטייט אז ער וועט נישט אפהיטן קיין היסטאריע, ער וועט נישט טריינירן דעם מאדעל דערמיט, און דער זכרון איז אויסגעלאשן. דריקט מען "continue", הייבט מען אן א שמועס אן אפהיטן קיין זכרון. אין דער לעקציע מאכט מען עס צו, ווייל מען וויל דא ניצן דעם זכרון.

מען קען אויך באניצן "custom instructions" כדי צוצופאסן ChatGPT, אבער דער קורס באמערקט אז די זענען אלגעמיין. דא וויל מען אז דער זכרון זאל ווערן אפגעהיטן און באניצט אין קומענדיגע שמועסן.

OpenAI דערמאנט אויך די פארבעסערונג פון זייערע פריוואטקייט- און זיכערקייט-סטאנדארטן, און אז team- און enterprise-קונים קענען ארבעטן מער עפעקטיוו. ביי דער ארבעט קען ChatGPT געדענקען דעם טאן, דאס קול און די פארמאט-פארליבטקייטן פונעם באניצער, ווי למשל וויאזוי מען שרייבט, וועלכע ווערטער מען באניצט, און ווי גלאט מען שרייבט ענגליש. פון זיין זכרון קען ער דאס ארויסלערנען און ארויסגעבן א רעזולטאט אין דעם זעלבן סטיל.

GPTs מיט זכרון און די אנוועדונגען (24:43:37)

אויך GPTs וועלן האבן זכרון. אינעם "Explore GPTs" טייל זעט מען GPTs וואס OpenAI אליין האט צוגעשטעלט: featured און trending, ווי למשל image generator און logo generator. מען קען אויך שאפן אן אייגענעם GPT דורכן "create" טייל. פון ChatGPT-ס אייגענעם טיעם זענען דא ,Data Analyst E·DALL און Creative Writing, און זיי אלע וועלן קענען אפהיטן זכרון. די דוגמא פון OpenAI: דער Books GPT העלפט געפינען דאס קומענדיגע בוך צו לייענען. ווען דער זכרון איז אנגעצונדן, געדענקט ער די פארליבטקייטן, ווי באליבטע זשאנערס אדער העכסטע ביכער, און פאסט צו די פארשלאגן אן דעם וואס מען זאל עס דארפן איבערחזר'ן.

די אנוועדונגען וואס ווערן דערמאנט:

- פראיעקט-פארוואלטונג: נאכפאלגן לויפנדיגע פראיעקטן, טערמינען (deadlines) און אחריות'ן;

- לערנען: ChatGPT געדענקט די לערן-צילן, למשל "איך וויל לערנען א שפראך אין אזויפיל וואכן, גיב מיר א road map". יעדן טאג וועט ער דערמאנען וועלכער טאג עס איז און וועלכע אויפגאבעס מען דארף פארענדיגן;
- פערזענליכע פארשלאגן: געדענקען וואס מען האט ליב ביי ביכער, פילמען אדער רעסטאראנען;
- געזונט און פיטנעס: נאכפאלגן פיטנעס-צילן, דיעטע-באגרעניצונגען און געניטונג-פארליבטקייטן, כדי צו געבן צוגעפאסטע עצות.

פראקטישע דעמא: פראיעקט-טערמינען און לערן-אינטערעסן (24:45:42)

מען שרייבט: "געדענק אז מיין פראיעקט-טערמין פאר דער מארקעטינג-קאמפיין מיטן נאמען Rise Up איז דעם 30סטן יוני." מען זעט "memory updated". מען גייט צו settings ← personalization ← manage memory, און דארט שטייט טאקע: "Project deadline for the marketing campaign named Rise Up is June 30th". אזוי וועט ער העלפן, אין וויטערדיגע שמועסן, נישט פארפאסן דעם טערמין, און אנפירן מיט די נויטיגע רעזולטאטן.

א צווייטער פראמפט: "איך בין אינטערעסירט צו לערנען וועגן renewable energy, האלט דאס אין זינען פאר אונזערע שמועסן." ווידער וועט ער זכרון דערהיינטיגט. מען באמערקט אז אין פראמפט איז געווען א שרייב-טעות, און דאך האט ער דעם זכרון ריכטיג דערהיינטיגט. פראמפטס ווי "keep that in mind" אדער "remember" לאזן ChatGPT אפהיטן שייכותדיגע פרטים וואס ער קען דערמאנען אין קומענדיגע שמועסן, און דאס מאכט די שמועס מער פערזענליך און מער באוואוסטזיניג אינעם קאנטעקסט.

נאך א פראיעקט-דוגמא און אויסמעקן זכרון (24:48:19)

מען שרייבט: "געדענק אז די קומענדיגע review-זיצונג פארן Rise Up פראיעקט איז באשטימט אויף 20סטן מאי, 3 אזייגער נאכמיטאג." ער דערהיינטיגט דעם זכרון ("memory updated "), און מען קען עס נאכקוקן אין settings ← personalization.

וויל מען אויסמעקן, קען מען דאס טון אויף צוויי אופנים:

- אויסמעקן א ספעציפישן זכרון גלייך פון "manage memory";
- אדער שרייבן א פראמפט, למשל: "איך האב דיר געזאגט צו געדענקען די review-זיצונג פארן Rise Up פראיעקט, מעק אויס יענעם זכרון, ווייל די זיצונג איז אפגעלייגט אדער אפגערופן." דעמאלט וועט ער אויסמעקן יענעם זכרון.

צום סוף זעט מען דעם ענטפער פון ChatGPT, וואס באשטעטיגט אז די קומענדיגע review-זיצונג פארן Rise Up פראיעקט איז באשטימט אויפן 20סטן מאי, 3 אזייגער נאכמיטאג.

ChatGPT זכרון און AI-ארבעטס-אויטאמאציע (24:49:50)

פראקטישע ביישפילן מיט דעם זכרון-פיטשער אין ChatGPT (24:49:50)

אין דעם טייל פון קורס וועט ווייטער געוויזן ווי אזוי מען באנוצט זיך מיט דעם זכרון (Memory) פיטשער פון ChatGPT. דער ביישפיל איז א פראיעקט מיטן נאמען "Rise Up". מען קען אנהייבן א שמועס, אדער מען גיט נאך פראמפטס. דער פראמפט וואס וועט געוויזן איז: "האלט אין זינען אז Sara איז די הויפט-דיזיינערין אויפן פראיעקט, און מאך זיכער זי איינצושליסן אין די רעלעוואנטע אפדעיטס."

עס ווערט ערקלערט אז אזעלכע פראמפטס וואס הייבן זיך אן מיט "keep in mind" (האלט אין זינען) זענען דער סימן פאר ChatGPT צו אפהיטן די אינפארמאציע אין זכרון. דא ווערט עס פארבונדן נאר מיטן Rise Up פראיעקט, אז Sara איז די הויפט-דיזיינערין דארט.

א בודזשעט-באריכט מיט א דעדליין (24:50:53)

דערנאך ווייזט דער קורס ווי מען קען בעטן ChatGPT ער זאל העלפן מיט באריכטן. מען קען אים געבן א XLSX טעקע (Excel). דער פראמפט וואס ווערט געשריבן: "איך דארף אריינגעבן דעם בודזשעט-באריכט פאר דער sustainability initiative) סביבה-פרעזערוואציע איניציאטיוו), נאר פאר דעם פראיעקט, ביז סוף קומענדיגן חודש." ערשט ווערט דערמאנט אז דער דעדליין איז דעם 20טן מאי און מען וויל עס פארטיג האבן פארן 19טן; אינעם פראמפט אליין ווערט שפעטער געשריבן דער 15טער מאי. עס ווערט באטאנט אז דאס איז דעם באניצערס אויסוואל, וואס מען וויל אז ChatGPT זאל געדענקען.

מען קען צולייגן: "לאמיר דאס דורכשמועסן אין די קומענדיגע שמועסן", אדער "איך וועל דיר געבן די Excel טעקע און דערנאך וועלן מיר אנהייבן צוגרייטן דעם באריכט", אדער פשוט "געדענק די זאכן". דער ענטפער וואס מען זעט: "פארשטאנען, דו דארפסט אריינגעבן דעם בודזשעט-באריכט פאר דער sustainability initiative פאר דעם Rise פראיעקט ביז דעם 15טן מאי. מיר קענען דאס דורכשמועסן אין אונזערע קומענדיגע שמועסן כדי מאכן זיכער אז דו ביסט אינגאנצן גרייט. אויב דו ווילסט אנהייבן מיט עפעס ספעציפיש, לאז מיך וויסן."

א וועג-מאפע ביזן דעדליין (24:52:26)

דער קומענדיגער פראמפט: "לאמיר אנהייבן מיטן בודזשעט-באריכט און מאכן א) road mapוועג-מאפע) אים צו פארענדיגן ביזן דעדליין." ChatGPT געדענקט שוין דעם דעדליין פונעם 15טן מאי און שטעלט אויס די טריט:

- זאמלען די דאטא
- אויסשטעלן די סטרוקטור פונעם באריכט
- שרייבן א ערשטן נוסח פונעם אינהאלט
- איבערקוקן און פארעכטן
- פארענדיגן דעם באריכט
- צוגרייטן פאר אריינגעבן
- אריינגעבן דעם באריכט

ביים ערשטן טריט, דאטא זאמלען, קען מען אריינלייגן די XLSX טעקע. דער קורס ערקלערט אז דאס איז א היפאטעטישער ביישפיל, אזוי אז קיין טעקע איז נישטא; אבער אז מען האט די טעקע, גיט מען זי איבער און ChatGPT הייבט אן אויסשטעלן דעם באריכט. מען קען אים זאגן וואס מען וויל זען: א dashboard, גראפיקן אדער אנאליז פון דער דאטא. מען זאגט אים וועלכע פונקציעס, dashboards אדער גראפיקן מען דארף, און ער הייבט אן שרייבן. אזוי ווערט דער זכרון אפגעהיט און באנוצט פאר יעדן צוועק.

צווייטער ביישפיל: געזונטע רעצעפטן אן ערד-ניס (24:53:59)

א צווייטער ביישפיל אין א נייעם שמועס: "איך וויל געזונטע רעצעפטן, ווייל איך האב אנגעהויבן מאכן געניטונגען, און געדענק אז איך בין אלערגיש צו peanuts ערד-ניס)." דער ציל איז אז ביי אלע רעצעפטן זאל ChatGPT געדענקען די אלערגיע און נישט אריינלייגן קיין ערד-ניס.

מען זעט אז ChatGPT ענטפערט און אפדעיטעט דעם זכרון. ער זאגט אז ס'איז גוט צו הערן אז מען קאנצענטרירט זיך אויף פיטנעס, און גיט רעצעפטן: א קינאָאַ און שווארצע בעבלעך סאלאט (quinoa and black bean salad), chicken and avocado wraps (הון מיט אוואקאדא), און veggie stir fry מיט tofu. איצט ווייסט ער אז מען איז אלערגיש צו ערד-ניס, און אפילו ביי רעצעפטן וואס זענען נישט ספעציפיש "געזונט" וועט ער עס געדענקען און נישט אריינלייגן קיין ערד-ניס.

צוזאמפאסונג און סעטינגס פונעם זכרון (24:55:01)

דער קורס פאסט צוזאם אז מיטן זכרון-פיטשער קענען team און enterprise קונים ארבעטן מער עפעקטיוו: ChatGPT לערנט זיך אויס דעם באניצערס סטיל און פרעפערענצן און בויט ווייטער אויף פריערדיגע שמועסן. עס ווערט דערמאנט אז עס איז דא א זכרון-אפטיילונג אויך פאר GPTs.

ווי אזוי צו קאנטראלירן:

- גיין צו Settings, דערנאך Personalization.
- דארט קען מען אויסלעשן דעם זכרון (Turn off) אויב מען וויל נישט אז ChatGPT זאל עפעס אפהיטן.
- מען קען אויסמעקן ספעציפישע זכרונות פון דארט.
- אדער מען שרייבט א פראמפט, למשל: "מעק אויס וואס איך האב דיר געזאגט וועגן דער אלערגיע צו ערד-ניס, ווייל איצט קען איך עסן ערד-ניס."
- עס איז אויך דא א Temporary Chat (צייטווייליגער שמועס), וואס היט נישט אפ קיין זכרון.

וואס איז AI ארבעטס-אויטאמאציע? (24:56:04)

דא הייבט זיך אן א נייער טעמע: AI workflow automation (AI ארבעטס-אויטאמאציע). עס ווערט ערקלערט אז אין דער היינטיגער שנעלער דיגיטאלער עקאנאמיע איז "busy work", די טרוקענע איבערגע ארבעט, א שטילער מערדער פון וואוקס. AI אויטאמאציע ערזעצט נישט סתם מענטשליכע ארבעט, נאר זי פארשטארקט זי, כדי מען זאל זיך קענען קאנצענטרירן אויף ארבעט וואס ברענגט די גרעסטע תועלת.

טראדיציאנעלע אויטאמאציע גייט לויט דער לאגיק "if this then that" (אויב דאס, דעמאלט יענץ). זי איז שטייף און ברעכט זיך ווען דער אינפוט ענדערט זיך אביסל. AI אויטאמאציע לייגט אריין AI, ווי LLMs) גרויסע שפראך-מאדעלן), אין די סדר-טריט, און דאס לאזט די סיסטעם קלערן, קאטעגאריזירן און שאפן אינהאלט אדער באשלוסן. אנשטאט נאר אריבערטראגן דאטא פון פונקט A צו פונקט B, קוקט AI אויף דער דאטא, פארשטייט וואס זי איז, און באשטימט דעם בעסטן קומענדיגן טריט.

א משל: טראדיציאנעלע אויטאמאציע איז ווי א conveyor belt (א באווועגלעכער פאס) וואס טראגט א קעסטל פון איין זייט צימער צו דער אנדערער. AI אויטאמאציע איז ווי א קלוגער סארטירער: ער עפנט דאס קעסטל, לייענט דעם לעיבל, באשליסט צי דער אינהאלט איז דרינגנד, און שרייבט א קורצע צוזאמפאסונג פארן מענעדזשער איידער ער לייגט עס אויף דער ריכטיגער פאליצע.

די מעלות פון AI אויטאמאציע (24:57:36)

• ריזיגע צייט-שפארונג: דורך באהאנדלען איבערגע מח-ארבעט, ווי שרייבן אימעילס, צווזאמפאסן מיטינגס און אריינשרייבן דאטא, קען AI צוריקגעווינען 10 ביז 20 שעה א וואך.

• ווייניגער מענטשליכע טעותים: AI ווערט נישט מיד אום 4 נאכמיטאג, און האלט א קאנסיסטענטן לעוועל פון געוויקייט ביים באארבעטן דאטא, אזוי אז copy-paste טעותים אדער פארפעלטע מעלדונגען ווערן א זאך פון דער פארגאנגענהייט.

• סקאלירבארקייט: מען קען באהאנדלען) leads 1,000מעגליכע קונים) אזוי גרינג ווי 10. קליינע טימס קענען ארויסגעבן די מאסע ארבעט פון א גרויסן טים אן צונעמען נאך מענטשן.

טראדיציאנעל קעגן AI-געטריבענע ארבעטס-פלוסן (24:58:06)

מען ווייזט א דיאגראמע וואס פארגלייכט די צוויי לויט פיר פונקטן:

• לאגיק-טיפ: טראדיציאנעל איז "אויב-דעמאלט", א שטרענג באדינגונג-באזירטער מאדעל. AI איז probabilistic) לויט מאגלעכקייטן): זי קאטעגאריזירט און קלערט, זי איז פלעקסיבל, און ס'איז דא א לערן-פראצעס ווייל מען באנוצט זיך מיט LLMs.

• אינפוט-טיפ: טראדיציאנעל ניצט סטרוקטורירטע דאטא, ווי spreadsheets. AI קען נעמען קולער, בילדער און אנדערע אומסטרוקטורירטע דאטא, וואס מאכט אין דער פראקטיק אסאך ארבעט גרינגער.

• באשלוס-מאכן: טראדיציאנעל איז בינאר, יא אדער ניין. AI ניצט קאנטעקסט: זי פאסט צווזאם, מאכט sentiment analysis) אנאליז פון שטימונג), א.א.וו.

• אויפהאלטונג: ביי טראדיציאנעל איז זי הויך, ווייל אלעס ברעכט זיך ווען דער פארמאט ענדערט זיך. ביי AI איז זי נידעריג, ווייל זי פאסט זיך צו פארשידענע טעקסטן און דאטא.

פאר וועמען איז דאס (25:00:11)

עס ווערט באטאנט אז AI אויטאמאציע איז נישט נאר פאר די טעק-ריזן; זי איז א "force multiplier" (א כח-פארמערער) פאר יעדן וואס פירט דיגיטאלע ארבעט:

• אנטרעפרענערס: אויטאמאטיזירן lead generation און אריינפירן נייע קונים, כדי זיך צו קאנצענטרירן אויף סטראטעגיע.

• טימס: גלאטיק מאכן אינערליכע קאמוניקאציע, פראיעקט-פארוואלטונג, אפדעיטס און טראנסקריפציע פון מיטינגס.

• מארקעטערס: שאפן פערזענליכע רעקלאמע-טעקסטן, פלאנירן סאשעל-מעדיע און אנאליזירן קונים-שטימונג אין גרויסן פארנעם.

• פראגראמירער: אויטאמאטיזירן קאוד-דאקומענטאציע, נאכפאלגן באגס און איבערגע טעסט-ציקלען.

אין 2026 האט זיך דער AI געצייג-פעלד איבערגעביטן פון פשוטע פארבינדערס צו אינטעליגענטע סיסטעמען. כדי צו בויען א קאנקורענץ-פעאיגן ארבעטס-פלוס, דארף מען פארשטיין די ארכיטעקטור-שטארקייט פון יעדן געצייג.

ערשטער געצייג: Zapier (25:01:14)

דער ערשטער געצייג איז Zapier, וועלכן מען קען באשרייבן ווי אן אינטואיטיוועך "orchestrator" (דיריגענט). Zapier איז דער באקאנטסטער נאמען אין אויטאמאציע און בלייבט דער אומבאשטריטענער קעניג פון פארבינדונגען, מיט איבער 8,000 integrations. ער ארבעט ווי אן אוניווערסאלער דאלמעטשער פארן אינטערנעט.

Zapier'ס גרעסטער שינוי איז אז זיי האבן אריינגעלייגט AI agents. אנשטאט א ליניארן A→B trigger, קען מען איצט בויען בלייבנדיגע AI אגענטן וואס לעבן איבער אלע אפליקאציעס. א ביישפיל: מען טריינירט אן אגענט אויף די PDF טעקעס פון דער פירמע און מען זאגט אים: ווען א קונה פרעגט א טעכנישע פראגע אין Slack, זוך אין אונזערע מאנואלן און שרייב א ענטפער; אבער שיק עס נאר ארויס אויב די שטימונג פונעם קונה איז נישט פראסטרירט.

Zapier'ס פלענער: Pro Trial און Teams (25:02:49)

עס ווערט געוויזן דער פרייז-בלאט פון Zapier. ס'איז דא א מאס פריי ניצן מיט געוויסע קרעדיטן. וועט עס עפנט א נייעם Zapier אקאונט ווערט אויטאמאטיש איינגעשריבן אין א 14-טאגיגן Pro Trial. דארט באקומט מען:

- פרעמיום אפליקאציעס, למשל פארבינדן מיט PayPal און Facebook Lead Ads
- webhooks, וואס זענען זייער אינטערעסאנט און וועלן שפעטער דורכגענומען ווערן
- Form AI fields

באגרעניצונגען: די task limit איז נאר 1,000, מען האט 30 טריט, און די פיטשערס auto replay און pay-per-task billing זענען נישט פאראן. אלע פרייע טרייאלס דויערן 14 טעג; דעם Pro Trial קען מען פארלענגערן גלייך פון Zapier אקאונט. פאר Team Trials דארף מען בעטן א פארלענגערונג דורך א לינק און זיך פארבינדן מיטן טים.

דער צווייטער אפציע איז Zapier for Teams: 1 מינוט trigger intervals, ביז 25 באניצער, user roles און permissions, shared app connections און א געטיילטער workspace. די task limit ביי Teams איז לויטן איצטיגן פלאן, 1,000 tasks, און import/export פון zaps איז נישט פאראן.

פינף אויטאמאציע-געצייג און א Zapier ביישפיל (25:04:23)

Zapier: פריי-טרייעל און ארויפגיין אויף א בעסערן פלאן (25:04:23)

דער קורס ענדיגט די באשרייבונג פון Zapier. מען זעט אז אין דער פרייער טרייעל-ווערסיע זענען געוויסע זאכן, ווי די signin אפציע, נישט צוטריטלעך, און מען קען פארלענגערן די טרייעל. אבער מען קען אויך גיין אויף די באצאלטע ווערסיעס, און ס'איז דא א געהעריגע דאקומענטאציע וועגן ארויפגיין אויף א בעסערן Zapier פלאן.

עס ווערט געגעבן א ביישפיל: אויב דער היינטיגער פלאן קאסט בערך $50 א חודש, קען מען ארויפגיין אויף $80 אינמיטן דעם באצאל-ציקל. אויב עס בלייבן נאך 15 טעג אינעם היינטיגן פלאן, באקומט מען $25 קרעדיט אויפן חשבון, און ביים ארויפגיין ווערט מען באלאסט מיט $55. דאס איז עפעס וואס קען מען אנווענדן פאר טימס, און עס ווענדט זיך פון אויפגאבע צו אויפגאבע, ווייל יעדע פירמע האט אנדערע אויטאמאציעס.

דער קורס רעקאמענדירט צו לייענען די דאקומענטאציע וועגן ענדערן דעם אינטערוואל פונעם באצאל-ציקל, ארונטערגיין אויף א נידעריגערן פלאן, אדער אפשטעלן דעם פלאן. Zapier ווערט אנגערופן א וואונדערליכער געצייג פאר אויטאמאציע, און עס שטייט אויפן ערשטן פלאץ אין דער רשימה.

דער צווייטער געצייג: make.com, דער "וויזועלער ארכיטעקט" (25:05:24)

דער צווייטער געצייג איז make.com, וואס ווערט אנגערופן "דער וויזועלער ארכיטעקט". עס ווערט ערקלערט: אויב Zapier איז פאר שנעלקייט, איז Make, וואס האט אמאל געהייסן Integromat, פאר געניקייט און פאר גרויסע פארנעמען. אין 2026 איז עס דער בעליבטסטער געצייג ביי אויטאמאציע-אינזשעניערן און אפעראציע-פירער וואס דארפן זען דעם גאנצן פראצעס.

עס ווערן געוויזן די לייזונגען וואס Make שטעלט צו, צעטיילט לויט געביטן:

- מארקעטינג
- פארקויפן (sales)
- אפעראציעס
- קונה-ערפארונג (customer experience)
- פינאנצן
- אינפארמאציע-טעכנאלאגיע
- מענטשן און ארבעטס-פלאץ
- פראדוקט

ווער עס וויל זיך מער אויסלערנען אדער זוכן וואס פאסט פאר זיינע באדערפענישן, קען זוכן אינעם פאסיגן געביט.

קעיס-סטאדיס, טעמפלעיטס און פרייזן פון Make (25:06:24)

Make גיט אויך קעיס-סטאדיס, למשל פאר אויטאמאציע פון סאושעל-מידיע. מען קען לייענען זייערע לייזונגען אין זייערע בלאגס, און זען מיט וועלכע אפליקאציעס זיי פארבינדן זיך, למשל Claude LLM, Browse AI, OpenAI ChatGPT און Facebook Pages.

מען קען אנהייבן מיט א פארטיגן טעמפלעיט. אלס ביישפיל ווערט געוויזן א טעמפלעיט וואס מאכט א קורצן אויסצוג פון דעם אינהאלט פון א וועבזייטל און שאפט דערפון א Facebook פאסט מיט ChatGPT און Browse AI. ס'זענען דא אסאך אנדערע אזעלכע טעמפלעיטס, און מען קען אויך מאכן אן אייגענעם workflow.

דערנאך ווערן דורכגעגאנגען די פרייזן. מען קען באצאלן א גאנץ יאר אויפאמאל אדער יעדן חודש, און די פלענער זענען געבויט אויף קרעדיטס א חודש:

- Free — אומזיסט: 1,000 קרעדיטס א חודש, אזוי ווי Zapier; no-code וויזועלע workflows; פארבינדונג מיט 3,000 אפליקאציעס; routers און filters; קונה-סופארט; און 15 מינוט צווישן איין לויף און דעם צווייטן.
- Core — $9 א חודש, די בעסטע ברירה פאר פרילענסערס וואס ווילן אויטאמאטיזירן זייערע workflows: אומבאגרעניצטע אקטיוויע scenarios, מער קאנטראל איבער צייט-באשטימטע scenarios, א העכערע גרענעץ פאר דאטא-טראנספער, און צוטריט צו דער Make API.

• א פלאן מיט פריאריטעט אין דער אויספירונג — $16 א חודש.

• Teams — בערך $29 א חודש. טימס קענען ניצן team roles , און מען קען שאפן און טיילן אייגענע scenarios.

• Custom pricing — פאר פירמעס וואס פירן זייער קריטישע געשעפטן און ווילן אויטאמאטיזירן זייערע געשעפט-פראצעסן: סופארט פאר custom functions, פארבינדונג מיט enterprise אפליקאציעס, enterprise סופארט 24/7, צוטריט צו א value engineering טים, און אוואנסירטע זיכערהייט-פונקציעס.

די מעגלעכקייטן און זיכערהייט פון Make (25:08:58)

עס ווערן אויסגערעכנט די פונקציעס פון Make. מען קען זיך פארבינדן מיט MCP servers, ארויסנעמען אינהאלט, זוכן אויפן וועב, מאכן אן אגענט און מאכן א toolkit. מען זעט ביישפילן פון workflow scenarios, למשל א no-code workflow builder, און מען קען ניצן system variables און custom variables.

וועגן פארבינדונגען: ס'זענען דא איבער 3,000 integrations, custom apps, און צוטריט צו enterprise אפליקאציעס וואס לויפן אויף די אייגענע סערווערס פון דער פירמע (on-premises). ביי דער אויספירונג פון workflows איז דא א מאקסימום צאל אקטיוע scenarios, scenarios וואס לויפן פאראלעל, א מאקסימום פייל-גרייס און א גרענעץ פאר דאטא-טראנספער. אונטער governance און observability זענען דא אומבאגרעניצטע באניצער, א grid, team roles, אנאליטיקס, און גוטע סופארט.

וועגן זיכערהייט ווערן דערמאנט two-factor authentication, single sign-on און domain claim . דאס hosting איז אויף AWS, און זייערע דאטא-צענטערס געפינען זיך אין צפון-אמעריקע און אין אייראפע (EU). דער קורס רעקאמענדירט צו אויספרובירן דעם געצייג.

דער דריטער געצייג: ChatGPT, דער "קאגניטיוועך מוח" (25:10:30)

דער דריטער געצייג איז ChatGPT. כמעט יעדער קען אים און ניצט אים אין זיין טאג-טעגלעכער ארבעט, און ער האט געמאכט א רעוואלוציע אין אויטאמאציע. דער קורס רופט אים "דער קאגניטיוועך מוח" פון 2026. ChatGPT איז געווארן פון א שמועס-אינטערפעיס א מוח וואס ארבעט קודם-כל דורך API פאר דער וועלט-עקאנאמיע. ער איז דער מאטאר וואס גיט דעם שכל און דאס טראכטן אינעווייניג אין די Zaps פון Zapier און די scenarios פון Make.

עס ווערן ערקלערט זיינע טיפע פארבינדונגען און מעגלעכקייטן:

• Function calling און ניצן געצייג: מען קען געבן ChatGPT רשות צו ניצן אנדערע געצייג. א ביישפיל: מען זאגט ChatGPT אז ער זאל קוקן אויף א קלאגע פון א קונה. אויב עס איז א טעכנישער באג, זאל ער ניצן דעם Jira געצייג צו עפענען א טיקעט. אויב עס איז א פראבלעם מיט באצאלן, זאל ער ניצן דעם Stripe געצייג צו זען די לעצטע באצאלונג פון דעם קונה.

• מולטימאדאלע מייסטערשאפט: ChatGPT קען זיין א ווייזועלער שריט אינעם workflow. למשל, א קונה שיקט א בילד פון א צובראכענעם חלק; דער AI קוקט אויף דעם בילד, דערקענט די נומער פונעם חלק לויט דעם וואס ער האט זיך אויסגעלערנט ביים טרייניר, און לאזט אויס א באשטעלונג אינעם אינווענטאר-סיסטעם.

וועגן מאדעלן: דערווייל איז דא ווערסיע 5.2, און מען קען אויך צוקומען צו געוויסע עלטערע (legacy) מאדעלן. אבער דער קורס רעקאמענדירט צו ניצן דעם לעצטן מאדעל, ווייל ער איז זייער אוואנסירט. די reasoning מאדעלן פון ChatGPT ווערן געוצט פאר אויטאמאציעס וואו עס ליגט אסאך אויפן שפיל. זיי טרעפן נישט בלויז דאס קומענדיגע ווארט, נאר זיי טראכטן דורך א פראבלעם. דעריבער פאסן זיי פאר אויטאמאטישן איבערקוקן לעגאלע קאנטראקטן אדער פאר קאמפליצירטע פינאנציעלע פאראויסזאגונגען.

ChatGPT איז גוט פאר שאפן טעקסט, אנאליזירן געפילן (sentiment analysis), ארויסנעמען דאטא פון אומארגאניזירטע מקורות ווי צומישטע אימעילס, און מאכן קאמפליצירטע באשלוסן.

דער פערטער געצייג: Notion AI (25:12:35)

דער פערטער געצייג איז Notion AI, א פארבונדענער ארבעטס-פלאץ. Notion האט זיך פארוואנדלט פון א געצייג פאר דאקומענטן און דאטא-באזעס אין אן אפערירנדע סיסטעם וואס איז פון יסוד געבויט אויף AI. לעצטנס האט מען צוגעגעבן אן AI אגענט. דער אגענט ווייסט אלעס וואס דער באניצער ווייסט, ווייל ער האט צוטריט צו דער גאנצער Slack היסטאריע און צו Google Drive דורך connectors. מען קען אים פרעגן, למשל: "וועלכע אויפגאבן בלייבן נאך אפן פון דעם פראיעקט וואס מיר האבן נעכטן באשפראכן אין Slack?" און ער מאכט א זייטל מיט א קורצן אויסצוג.

א פונקציע פון "שטעל איין און פארגעס": מען מאכט א דאטא-באזע פאר מיטינג-נאטיצן און מען לייגט צו אן AI אייגנשאפט וואס הייסט "action item". יעדעס מאל וואס א מיטינג ענדיגט זיך, לייענט Notion AI אויטאמאטיש דעם טראנסקריפט און פילט אויס די action items, אן דעם וואס מען זאל דארפן קליקן א קנעפל.

נאך א פונקציע איז Q&A פאר טימס: Notion AI ארבעט ווי א ביבליאטעקאר 24 שעה א טאג. אנשטאט אז אן ארבעטער זאל פרעגן אין Slack וואו מען געפינט די PTO פאליסי, פרעגט ער Notion AI, און עס גיט אים דעם ענויעם דאקומענט און דעם געהעריגן אפטייל. דער געצייג פאסט בעסטנס פאר אינערלעכן וויסן-פארוואלטונג, נאכפאלגן פראיעקטן, און טימס וואס ווילן אז זייער דאקומענטאציע זאל זיין לעבעדיג און זיך אליין ארגאניזירן.

די פרייזן:

- Free — $0 א חודש, מיט געוויסע פונקציעס.
- א פלאן פון $10 א חודש — אומבאגרעניצטע blocks פאר צוזאמענארבעט, אומבאגרעניצטע פייל-ארויפלאדונג, custom forms א.א.וו.
- Business — מיט Notion AI, SAML SSO, granular database פערמישאנס, באשטעטיגן (verify) זייטלעך, conditional forms logic א.א.וו.
- Enterprise — פארבינדונגען מיט אסאך אנדערע געצייג, מיט זייערע פונקציעס.

דער פינפטער געצייג: Pabbly (25:14:38)

דער פינפטער געצייג איז Pabbly. ער האט זיך אויסגעהאקט א גרויסן פלאץ אינעם מארק, ווייל ער איז דער ביליגסטער. ער איז אן אלטערנאטיוו צו Zapier, וואס צילט ספעציעל קעגן דעם "SaaS טאקס" וואס פרעסט אויף דעם פראפיט פון קליינע געשעפטן.

פארוואס איז ער באזונדער?

- מען באצאלט נישט פאר אינערלעכע אויפגאבן. למשל, אין Zapier ווערט א filter אדער א data formatter געראכנט ווי א task וואס קאסט געלט; אין Pabbly זענען זיי אומזיסט. מען באצאלט נאר פאר דער לעצטער אקציע, ווי למשל שיקן אן אימעיל.
- לעבנסלאנגע דילס: Pabbly גיט אפט אן איינמאליגע באצאלונג פאר לעבנסלאנגע אויטאמאציע. פאר א סטארטאפ קען דאס אויסשפארן בערך $2,000 א יאר.
- גרויסע פארלעסלעכקייט פאר מארקעטינג: Pabbly האט טיפע פארבינדונגען מיט אומגעוועניליכע מארקעטינג-געצייג, ווי mailers און SMS gateways, וואס Zapier באגרעניצט אמאל אדער רעכנט פאר זיי טייער.

Pabbly פאסט פאר אנטרעפרענערן וואס היטן אויפן בודזשעט, מארקעטינג-אגענטורן וואס פירן א גרויסע מאס lead generation, און באניצער וואס האסן מאנאטליכע אבאנעמענטן.

א פראקטישער ביישפיל: בויען א workflow אין Zapier (25:16:14)

יעצט ווערט געוויזן ווי אזוי מען בויט אן אויטאמאציע-workflow מיט Zapier. מען גייט צום אפיציעלן וועבזייטל zapier.com. אויפן home זייטל זעט מען אסאך workflows וואס זענען שוין געמאכט געווארן. מען קליקט אויף create, גיט א נאמען (למשל "news app"), און מען זעט דעם workflow.

עס זענען דא צוויי וועגן: מען קען קליקן אויף trigger און בויען דעם workflow אליין מיט triggers און actions, אדער מען קען באשרייבן דעם workflow פאר דעם copilot. אין דער לעקציע ווערט געשריבן אזא פראמפט: מען זאל שאפן א workflow אויטאמאציע מיט Google Sheets, וואו די דאטא קומט פון א Google Sheet וואס האט די קאלאמען name, email און source. דער source קען זיין LinkedIn, Twitter, Instagram אדער Facebook. leads פון Instagram, Twitter אדער Facebook זאלן געראכנט ווערן ווי leads מיט הויכע פריאריטעט, און מען זאל שיקן אן אימעיל לויט דעם. מען לייגט אריין די אייגענע אימעיל-אדרעס.

דער רעזולטאט איז א workflow פון דריי שריט:

- ערשטנס, דער trigger פון Google Sheets.
- צווייטנס, Zapier AI צו באצייכענען יעדן lead ווי הויכע אדער נידעריגע פריאריטעט.
- דריטנס, שיקן דעם אימעיל.

Zapier-אגענטן, באשלוסן און Webhooks (25:18:23)

דער אגענט פירט אויס דעם Zap-ווארקפלאו (25:18:23)

מען הייבט אָן מיט א פראמפט צום Copilot פון Zapier, אז א בליצבריוו (Email) זאָל ווערן ארויסגעשיקט יעדע 15 מינוט צו דער אקטיוואר אדרעס. נאכן אריינגעבן די פרטים זעט מען ווי דער גאנצער ווארקפלאו ווערט אויטאמאטיש אויסגעשטעלט דורכן אגענט אליין. מען דאַרף נאָר אביסל ווארטן ביז ער ענדיקט.

דערנאך גייט מען איבער די טריט וואָס דער אגענט טוט:

- ער לייגט צו די Zap-טריט ("add the zap steps") און קלייבט אויס אלע פיר טריט.
- ער נעמט די פעלדער ("get the fields") און דערהיינטיקט די געגעבענע Zap-טריט.

• ער לאזט לויפן (trigger) אלע טריט, פילט אויס און אנאליזירט די זאכן אויטאמאטיש, און גיט צוריק די דאטא.

דער קורס באטאָנט אז דאָס איז די גרינגסטע זאך וואָס מען קען דאָ טון. ביים דריטן טריט זעט מען אז אינעם קערפער (body) פון דעם בריוו שטייט "new or high priority lead detected", מיט אלע צוגעהעריקע פרטים. מען קען אויך אריינלייגן טעקסט אדער דאטא, און מען קען אנצינדן "read receipts" (באשטעטיגונג אז דער בריוו איז געליינט געוואָרן), אָבער אינעם ביישפיל לאזט מען עס אזוי.

נאכדעם פרובירט מען דעם טריט (test), און ער ווערט אויך פארענדיקט. מען קוקט אריין אין דער Email-קעסטל און זעט אז דער בריוו איז טאקע אנגעקומען. עס ווערט געזאגט אז Zapier האט א וואונדערליכע סיסטעם פון אגענטן, און אז דאָס איז די רעוואלוציע וואָס קומט אין דער AI-וועלט, וואָס וועט אויטאמאטיזירן א סך זאכן.

איבערחזרן דעם וואָרקפלאו און אריסגעבן (25:20:27)

דאָס איז געווען איין וועג צו אויטאמאטיזירן די זאך מיט ChatGPT. ווען דער וואָרקפלאו איז פארטיג, קען מען אים אריסגעבן (publish), און דאַן גייט ער "live". מען חזרט איבער די פשוטע טריט:

• ערשטנס, ליינען די ספּרעדשיט. אינעם ביישפיל הייסט זי "untitled" און עס איז צוגעבונדן א טאבעלע מיט דאטא.

• צווייטנס, Zapier AI אנאליזירט די דאטא און גיט זי צוריק.

• דריטנס, מען שיקט ארויס דעם בריוו (outbound mail).

• לויטן אנאליז פון דער פריאריטעט מעלדט עס צי דער) Leadא מעגליכער קונה) איז פון נידעריקער אדער הויכער פריאריטעט.

עס ווערט ערקלערט אז מיט אגענטן קען מען גרינג שאפן אויטאמאטישע וואָרקפלאוס אָן קיין גרויסע ידיעות; די אגענטן וועלן אויספירן די ארבעט. מען מוז אָבער שרייבן א זייער אויספירלעכן פראמפט, ווייל דאָס איז דער שליסל צו באקומען די ארבעט געטון.

באשלוסן און באדינגונג-לאָגיק מיט ChatGPT (25:21:30)

דער נעקסטער טייל באהאנדלט ווי מען מאכט באשלוסן און באדינגונג-לאָגיק, כדי ChatGPT זאָל מאכן קלוגע באשלוסן. דער עיקר: מען נעמט די פראגעס פון באניצער אדער אריינגעגעבענע דאטא, מען פאַראַרבעט זיי מיטן AI-מאדעל, און דערנאך באשטימט מען וואָס פאר א קלוגע אקציע צו טון.

דער סצענאַר איז קלאסיפיצירן Leads. ס'איז דא א פאָרם וואו מעגליכע קונים גיבן איין זייערע פרטים, און לויט זייערע ענטפערס קען ChatGPT זיי איינטיילן אין הויכן, מיטלסטן אדער נידעריקן אינטערעס. א ביישפיל פון אזא אנפראג: "איך דארף א לייזונג וואָס זאָל מיר אויטאמאטיזירן מיינע סאשעל-מידיע קאמפיינס. וויפיל וועט עס קאָסטן פאר א טים פון פינף?" דערנאך, אויב דער Lead פרעגט "קענסטו מיר דערציילן מער וועגן די פראדוקט-אייגנשאפטן?", ענטפערט ChatGPT מיט "א דאנק פארן זיך פארבינדן" לויטן אינטערעס, און עס געפינט די פאסיגע פראדוקט-קאטעגאריע.

מען פירט דורך די סימולאציע: מען גיט איין דעם ערשטן פראמפט, און ChatGPT הייבט אָן געבן לייזונגען, צ.ב. Hootsuite, Sprout Social און Buffer Team. דערנאך טייפט מען דעם צווייטן פראמפט, "can you tell more about the product features", און ChatGPT גיט לייזונגען לויט די געזאגטע מכשירים.

צופאסן א פראמפט מיט "if/else"-לאָגיק (25:23:36)

דערנאך ווייזט מען ווי צו צופאסן א פראמפט פאר א ספעציפישע לייזונג. נעמט אז מען בויט אן AI-אגענט, און ס'רוב קונים לייגן איין טענות אויפן פאָרטאַל אז זיי האבן פראבלעמען מיט באצאָלן. מען גיט צו GPT די פרטים פון דער פראגע (צ.ב. "איך בין אינטערעסירט צו לערנען מער וועגן אייערע פראדוקט-אייגנשאפטן"), און מען דארף אים אָנזאָגן ווי צו באשליסן. דער פראמפט זעט אויס בערך אזוי:

- אויב די פראגע אנטהאלט דאָס וואָרט "payment" אדער "billing", ענטפער: "א דאנק פארן זיך פארבינדן. עס זעט אויס אז איר האט א פראבלעם מיט רעכענונגען; מיר וועלן אייך גלייך פארבינדן מיט אונזער פינאנץ-טים."
- אַנדערש, אויב די פראגע אנטהאלט דאָס וואָרט "features", ענטפער: "א דאנק פאר אייער אינטערעס, אָט איז מער אינפארמאציע וועגן אונזערע פראדוקט-אייגנשאפטן."
- אין אלע אנדערע פעלער: "א דאנק פארן זיך פארבינדן, אונזער סופּאָרט-טים וועט זיך באלד צוריקמעלדן."

דער קורס באמערקט אז דאָס איז פשוט די פראגראמירער-לאָגיק פון "if" און "else" וואָס מען לערנט. אזוי קען מען דורכן פראמפט שאפן אן "if/else"-באדינגונג, און דאָס זעלבע קען מען אריינלייגן אין Zapier, אזוי ווי מען האט פריער געטון.

א ביישפיל פון דער פראקטיק: Lead Generation (25:25:06)

Lead Generation איז דער פראצעס פון כאפן מעגליכע קונים און זיי פארוואנדלען אין באמת'ע באצאָלנדע קונים. דאָס איז א זייער וויכטיגע ארבעט פאר פארקויף-טימס, באזונדערס אין אינדוסטריעס ווי Real Estate, SaaS און E-commerce. די Leads קומען געוועהנליך פון וועבזייטלעך, איינגעבענע פאָרמס און "landing pages".

מיט Zapier קען מען אויטאמאטיזירן די אנכאפונג פון דער Lead-דאטא פון א פאָרם, זי האלטן אין Google Sheets, און מיט Zapier אויטאמאטיש איינטיילן די Leads און אנצינדן די פאסיגע וווייטערדיגע שריט:

- דער טריגער: א נייע איינגעגעבענע פאָרם, צ.ב. פון Google Forms, מיט נאָמען, Email, פירמע אדער אינדוסטריע. די פאָרם קען אויך אנטהאלטן א Lead-קוואליפיקאציע: אינטערעסירט, נישט אינטערעסירט, אדער "מעלד זיך שפעטער". יעדעס מאָל וואָס א מעגליכער קונה פילט אויס די פאָרם, באמערקט Zapier דאָס און הייבט אָן דעם וואָרקפלאו.
- די אקציע: האלטן די דאטא אין Google Sheets. יעדע איינגעבונג שאפט א נייע שורה מיט די פרטים אין ארגאניזירטע קאָלומנעס: נאָמען, Email-אדרעס, נאָמען פון דער פירמע, אינדוסטריע. מען קען אויך דערהיינטיקן דעם סטאטוס פון דעם Lead.

אזוי ווערט די Google Sheet א שיינע דאטא-באזע פאר פירן Leads, וואָס גיט דעם פארקויף-טים א גרינגע איבערבליק פון מעגליכע קונים. מען קען אויך אנפירן וווייטערדיגע שריט און איינטיילן די Leads. Zapier האט איינגעבויטע פילטערס און וואָרקפלאוס וואָס מען קען איינשטעלן צו קאטעגאריזירן Leads לויט זייערע ענטפערס. צ.ב. Leads וואָס זענען באצייכנט אלס "interested" גייען אריין אין א ליסטע פון הויכער פריאריטעט, און די איבעריגע אין אן אלגעמיינע ליסטע פאר וווייטערדיגע באהאנדלונג. די איינטיילונג גיט דעם פארקויף-טים א מעגליכקייט צו שטעלן זייערע כוחות אויפן ריכטיגן אָרט.

Webhooks, פילטערס און Email-סופֿאָרט (25:27:39)

דערנאך ווייזט מען נאך עטליכע זאכן אין Zapier: Webhook, פילטערירן, און Email-סופֿאָרט. מען קען זיך באניצן מיט דער אפיציעלער דאקומענטאציע פון דעם API. דאָרט זעט מען, צ.ב., דאטא אין "form encoded"-פֿאָרעם (ערשטער נאָמען און לעצטער נאָמען), אדער דאטא אין JSON-פאָרמאט.

א Webhook איז עפעס ווי א URL, און וועט מען שאפט א Webhook קען מען זיך באניצן מיט דעם URL. כדי זיך צו באניצן מיט א Zapier-Webhook, גייט מען אין Google Sheets, אין "Apps Script", און מען שרייבט דאָרט א JavaScript-פונקציע. אין איר שאפט מען א ווערִיאבל "url", וואָס האלט דעם Zapier-Webhook. כדי צו שאפן דעם Webhook אליין, קליקט מען אויף דעם צוגעהעריגן "event"; מען דארף דורכגיין א ריכטיגע איינשטעלונג. ערשט שאפט מען א טריגער, און אינעם "configuration"-אָפּטייל שטעלט מען איין די פאסיגע זאכן.

דער Copilot בויט א Webhook-וואָרקפלאו (25:29:12)

אנשטאט עס אליין צו טון, קען מען בעטן דעם Copilot, למשל: "שאף א Webhook מיט א טריגער-אקציע, מיט פילטער-באדינגונגען פאר Leads פון הויכער ביז נידעריקער פריאריטעט". מען קען אים אויך בעטן א וואָרקפלאו פון דריי טריט: א Lead-איינטיילונג, שיקן א בריוו, און צולעצט נאכאמאל שיקן א בריוו; מען גיט איין די אייגענע Email-אדרעס.

דער אגענט הייבט אָן ארבעטן און שאפט זיין אייגענעם וואָרקפלאו. דאָס קען נעמען א שטיק צייט. ערשט נעמט ער אראפ וואָס ס'איז דא, דערנאך שאפט ער דעם טריגער אלס ערשטן טריט, און דאַן הייבט ער אָן צולייגן נייע טריט. מען זעט אז ער שאפט קודם דעם Webhook. אינעם "configure"-אָפּטייל קלייבט ער אויס א "child key" און גייט ווייטער. מען קען אויך איינשטעלן די אפליקאציע-קאנפיגוראציע מיט "Filter by Zapier". עס ווערט באטאָנט אז מען דארף גארנישט טון; אלעס ווערט באזאָרגט דורכן אגענט, און וועט דער ערשטער טריט איז פארטיג, בויט ער דעם וואָרקפלאו.

דער קורס רעקאמענדירט צו באטראכטן דעם באצאָלטן ווערסיע פון Zapier, ווייל ס'רוב אויטאמאציע-ארבעט קען מען דערמיט גרינג אויספירן.

א לעצטער עצה וועגן פראמפטס (25:30:44)

צום סוף פון דער לעקציע קומט א וויכטיגע עצה: וועט מען שרייבט א פראמפט, זאָל מען געבן א טיפערן געדאנק און אן אויספירלעכע באשרייבונג, און די אגענטן וועלן ארבעטן אויף דעם אופן. מען שליסט אָפּ מיט דער באמערקונג אז Zapier קען גרינג אריינברענגען GPT , און אז אמאָל וועט עס אויטאמאטיש דערקענען דעם OpenAI-חשבון.

OpenAI o1: א מאדעל וואס טראכט (25:31:45)

סוף פון א פריערדיגער דעמא (25:31:45)

דער קורס ענדיגט א פריערדיגע דעמא: מען דארף נאר אריינשרייבן די אימעיל-אדרעס, און מען זעט ווי דאס געצייג הייבט אן צולייגן די באטרעפנדע זאכן. דערנאך קליקט מען "continue", און עס הייבט אן טעסטן, און מען קען אויך טעסטן די באזונדערע טריט. עס ווערט באצייכנט אלס א שטארקע פונקציע וואס מען קען אויספרובירן.

דערנאך הייבט זיך אן א נייער טייל וועגן דעם OpenAI מאדעל o1. דער פלאן פון דעם טייל: וואס מאכט o1 אזוי באזונדער, ווי אזוי עס קלערט דורך (reasoning), וואו עס איז נוצליך, און פארוואס עס איז וויכטיג צו לערנען וועגן דעם מאדעל אין דער צייט פון "קלוגע מאשינען".

וואס איז o1? (25:32:19)

o1 איז OpenAI'ס נייער AI מאדעל, אבער לויט דעם קורס איז עס נישט פשוט א קליגערער chatbot, נאר א "מאשין פון דורכקלערן" (reasoning engine). מאדעלן ווי GPT-4 זענען גלענצנד אין שפראך און שעפערישקייט, אבער o1 גייט א טריט ווייטער: עס קען טראכטן לאגיש, לייזן פראבלעמען מיט עטליכע טריט פון שכל, און אנאליזירן קאמפליצירטע דאטא. בקיצור, עס ווערט געזאגט או o1 איז דאס נאענטסטע וואס מען איז געקומען צו בויען אן AI וואס קען דורכטראכטן א ארבעט, און נישט נאר ענטפערן.

פארוואס איז דאס וויכטיג? ווייל פראבלעמען אין דער אמת'ער וועלט, אין מאטעמאטיק, וויסנשאפט, קאוד און אינעם לעבן, פאדערן א געארדנטע מחשבה, נישט נאר שנעלע ענטפערס.

וואס הייסט "דורכקלערן" און פארוואס דארף מען עס? (25:32:51)

איידער מען פארשטייט ווי o1 ארבעט, פרעגט דער קורס: וואס איז reasoning? עס איז דער פראצעס פון אנאליזירן אינפארמאציע, געפינען פארבינדונגען און ארויסנעמען מסקנות, ווי ביים לייזן א רעטעניש.

דער ביישפיל: מען זעט טונקעלע וואלקנס, און מענטשן טראגן שירעמס. אפילו אויב עס רעגנט נאך נישט, קלערט מען אז עס גייט באלד רעגענען. AI דארף די זעלבע פעאיקייט. אן reasoning איז אן AI ווי א פאפוגיי, וואס חזר'ט נאר איבער; מיט reasoning ווערט עס איינער וואס לייזט פראבלעמען, און דאס איז פונקט וואס o1 ברענגט.

ווי אזוי o1 קלערט דורך: דריי מעכאניזמען (25:33:55)

- Chain of thought) קייט פון געדאנקען): o1 לייזט א פראבלעם טריט ביי טריט, ווי מען טוט אויף א פאפיר. ביישפיל: וויפיל איז 12 מאל 15? o1 זאגט נישט גלייך 180, נאר עס רעכנט: 10 מאל 15 איז 150, דערנאך 2 מאל 15 איז 30, און דערנאך לייגט עס צוזאמען: 150 + 30 = 180. (אין די אויטאמאטישע קעפשאנס שטייט א טעות "115", אבער דער רעזולטאט איז 150.) די מעטאדע פארמינערט טעותים און ווייזט די לאגיק הינטער יעדן ענטפער.
- Reinforcement learning) לערנען דורך באלוינונג): o1 ווערט בעסער מיט דער צייט דורך לערנען פון הצלחות און דורכפאלן, ווי א מענטש. אויב איין מעטאדע ארבעט נישט, פרובירט עס אן אנדערע, זעט דעם רעזולטאט און פארבעסערט זיך דאס נעקסטע מאל. עס איז ווי א שאך-שפילער וואס ווערט שארפער מיט יעדער שפיל.
- Reasoning tokens: ווען o1 לייזט א קאמפליצירטע ארבעט, באניצט עס זיך מיט אומזעעדיגע "טאקענס" צו האלטן חשבון פון זיינע געדאנקען, ווי פאפירלעך אויפן קאפ (sticky notes). ווי א מענטש וואס לייזט א חשבון און באקריצלט די טריט אויף א פאפיר, אזוי טוט o1 אינעווייניג, און דאס העלפט אים בלייבן געארדנט און געוי.

ביישפילן פון o1 אין אקציע (25:35:26)

ערשטער ביישפיל, א חשבון: וויפיל איז 25 + 37? o1 טראכט: 20 + 30 = 50, 5 + 7 = 12, 50 + 12 = 62, און דער ענדגילטיגער ענטפער איז 62. דער קורס באטאנט: עס איז נישט קיין כישוף, נאר מעטאדישע דורכקלערונג, און דאס מאכט o1 מער פארלעסלעך און דורכזיכטיג.

צווייטער ביישפיל, אויסבעסערן קאוד (debugging): אויב מען פרעגט "פארוואס גיט מיין Python קאוד א טעות?", טוט o1 אזוי:

- עס לייענט דעם קאוד שורה נאך שורה;
- עס געפינט וואו דער טעות פאסירט;
- עס ערקלערט פארוואס ער איז פארגעקומען;
- עס שלאגט פאר א תיקון, טריט ביי טריט.

עס איז ווי א רואיגער, לאגישער שותף צום קאודירן וואס ערקלערט שטענדיג זיין מחשבה.

וואס מאכט o1 אנדערש: א פארגלייך מיט GPT-4 (25:36:26)

דער קורס ווייזט א קורצן פארגלייך:

- פאראויסזאגן דאס נעקסטע ווארט: ביידע, GPT-4 און o1, טוען עס.
- דורכטראכטן טריט ביי טריט: GPT-4 נישט, o1 יא.
- זיך אליין פארריכטן: ביי GPT-4 א פשוטע מדרגה, ביי o1 א זייער פארגעשריטענע.
- לייזן קאמפליצירטע פראבלעמען: GPT-4 טייל מאל, o1 שטענדיג.
- דורכזיכטיגע דורכקלערונג: GPT-4 האט נישט, o1 האט.

וואו מען קען באנוצן o1 (25:36:57)

צוליב איר געארדנטער דורכקלערונג איז o1 נוצליך אין עטליכע געביטן:

- מאטעמאטיק און וויסנשאפט: לייזן גלייכונגען, ערקלערן עקספערימענטן.
- קאודירן: שרייבן, אויסבעסערן און פארשטיין פראגראמען.
- פארשונג: אנאליזירן דאטא, צוזאמענפאסן ארטיקלען און באשטעטיגן פאקטן.
- שעפערישע ארבעט: פלאנירן אינהאלט, לאגיש אויסטראכטן אידייען (brainstorming).
- אנאליז פון בילדער: אויסטייטשן בילדער טריט ביי טריט.

וואו מען דארף טראכטן, ברענגט o1 קלארקייט.

געווייגטלעכע טעותים וואס o1 פארמיידט (25:37:29)

עלטערע מאדעלן האבן געהאט עטליכע געווייגטלעכע פראבלעמען: זיי זענען גלייך געשפרונגען צו מסקנות, האבן איגנארירט קליינע פרטים, און האבן איבערגעחזר'ט טעותים אין נייע צוזאמענהאנגען. o1 פארמיידט דאס אלץ דורכן דורכקלערן אין טריט, באשטעטיגן איר אייגענע לאגיק, און לערנען פון רעאקציעס. דערפאר איז עס מער פארלעסלעך, בפרט וען עס שטייט אסאך אויפן שפיל.

פארוואס o1 איז ווייכטיג, און סך-הכל (25:38:03)

דער קורס זאגט אז AI וואס קלערט דורך איז די צוקונפט, און זי איז שוין דא. צו לערנען ווי o1 ארבעט העלפט: באנוצן AI מער עפעקטיוו, שטעלן בעסערע פראגעס, פארשטיין ווי מאשינען טראכטן, בלייבן פאראויס אין טעכנישע פארשונג און חינוך, און אליין ווערן א קריטישער דענקער. עס איז נישט נאר וועגן AI, נאר וועגן זיך גרייטן צו א וועלט וואו טראכטן מיט AI איז א טייל פון ווי מען ארבעט, לערנט און שאפט.

סך-הכל: o1 ענטפערט נישט נאר, נאר קלערט דורך; עס לערנט פון טעותים און פארבעסערט זיך; עס ערקלערט איר לאגיק טריט ביי טריט; עס איז גענוי, ערקלערבאר און נוצליך אין פארשידענע אינדוסטריעס; און עס איז די צוקונפט פון קלוגע סיסטעמען. צי מען איז א סטודענט, א פראגראמירער, אן אנאליטיקער אדער פשוט נייגעריג, איז עס א געצייג וואס העלפט בעסער טראכטן.

צווייטער מאדול: פראמפטן אין o1 מאדעלן (25:39:35)

דער צווייטער מאדול באהאנדלט ווי מען שרייבט פראמפטס פאר o1 מאדעלן. א פראמפט איז דער אופן ווי מען רעדט מיט AI: מען גיט אנווייזונגען און באקומט ענטפערס. לויט דעם קורס זענען די זאכן געווארן פשוטער מיט די נייע מאדעלן (עס ווערט דערמאנט GPT-4o): זיי פארשטייען פראמפטס געניעער, אזוי אז מען קען זיין קלאר, קורץ און דירעקט אן איבער-ערקלערן.

פיר פרינציפן פון פראמפטן אין o1:

- זיין פשוט און דירעקט.
- מען דארף נישט אויסדריקלעך בעטן א chain of thought; די דורכקלערונג איז שוין איינגעבויט.
- סטרוקטור איבער באשרייבונג: גיבן א סטרוקטור, נישט לאנגע באשרייבונגען.
- ווייזן, נישט זאגן: ווייזן דעם טאן מיט ביישפילן, נישט נאר דערציילן.

די פרינציפן העלפן באקומען בעסערע און שנעלערע רעזולטאטן, גלאט און עפעקטיוו.

דעמא: אויפשטעלן די סביבה (25:40:37)

דער ערשטער קאוד-בלאק טוט צוויי זאכן: ער שטילט ווארנונגען און באקומט דעם OpenAI API שליסל.

- import warnings: ברענגט אריין Python'ס איינגעבויטן מאדול פאר ווארנונגען, מעלדונגען וועגן מעגלעך פראבלעמאטישע אדער אומגעריכטע געשעענישן אין קאוד, וואס העלפן נאכגיין קלייניע פראבלעמען אן אפשטעלן דעם פראגראם.
-)"ignore"warnings.filterwarnings(: הייסט Python איגנאריין אלע ווארנונגען.
- from helper import get_openai_api_key: ברענגט אריין א פונקציע פון א מאדול "helper", וואס ברענגט דעם שליסל וואס מען דארף צו ארבעטן מיט OpenAI'ס סערוויסעס.
- openai_api_key = ()get_openai_api_key: רופט די פונקציע און היט דעם שליסל אין א וואריאבל, צו מאכן באשטעטיגטע API בקשות.

דער צווייטער בלאק:

- import json: צו ארבעטן מיט JSON דאטא, געווינטלעך ביי API.
- from IPython.display import display, Markdown, HTML: פונקציעס צו ווייזן אינהאלט.

- from openai import OpenAI: די קלאס צו ארבעטן מיט OpenAI.
- OpenAI(api_key=openai_api_key) = client: שאפט א קליענט מיט דעם שליסל.
- GPT_MODEL = "gpt-4o-mini" און O1_MODEL = "o1-mini": די נעמען פון צוויי מאדעלן פאר די בקשות.

דעמא: א "שלעכטער" פראמפט פאר אינסולין (25:43:43)

מען שאפט א וואריאבל bad_prompt , א טעקסט וואס בעט דעם מאדעל צו שרייבן א פונקציע וואס גיט ארויס SMILES IDs) וועג צו פארשטעלן א מאלעקולארע סטרוקטור) פאר אלע מאלעקולן אין אינסולין. דער פראמפט פירט אויך דעם מאדעל צו טראכטן טריט ביי טריט: אידענטיפיצירן די מאלעקולן, שאפן די פונקציע, און דורכגיין יעדן מאלעקול צו ארויסגעבן זיין SMILES.

דערנאך שיקט מען עס מיט client.chat.completions.create, מיטן מאדעל O1_MODEL; דער messages פאראמעטער שיקט דעם bad_prompt אונטער דער user ראלע.

צום וויזן דעם רעזולטאט אין א Jupyter notebook:

- display(HTML(...)): א div מיט א העל הינטערגרונט און א ראנד, מיט א קעפל "Markdown Output" מיט אן עמאדזשי פון א פייל אראפ, און hr-טעגס פאר האריזאנטאלע שורות פאר און נאכן קעפל.
- display(Markdown(...)): נעמט דעם אינהאלט פונעם ענטפער און וויזט עס פארמאטירט: טעקסט, קעפלעך, רשימות.

ביים לויפן זעט מען דעם ענטפער: טריט 1, אידענטיפיצירן די מאלעקולן אין אינסולין, chain A און chain B; טריט 2, צוגרייטן SMILES סטרינגס פאר יעדער אמינא-זייערע; דערנאך שאפן די פונקציע.

צום סוף ווערט צוגעגרייט א "good prompt" ("Generate a function that outputs a SMILES ID..."), ווייל דער פריערדיגער איז געווען א שלעכטער.

סטרוקטורירטע פראמפטס און פלאנירן מיט o1 (25:47:21)

א גוטער פראמפט: א קונה-סערוויס אגענט פאר AnyCorp (25:47:21)

דער קורס האלט ביי צוגרייטן א גוטן פראמפט, און מען דרוקט אים אויס כדי מען זאל זען דעם חילוק קלאר. עס ווערט באטאנט אז דער קאוד זעט אויס גרויס, אבער מען דארף נישט ווערן איבערגעשטרענגט, ווייל עס ווערט צעלייגט שריט נאך שריט.

דאס ביישפיל איז א סטרוקטורירטער פראמפט וואס פירט אן AI-אסיסטענט צו פירן זיך ווי א קונה-סערוויס פארשטייער פאר א פירמע מיטן נאמען AnyCorp, וואס פארקויפט לאגער-לעזונגען (storage solutions), למשל קאסטנס (bins). דער ציל פונעם פראמפט איז צו זיכער מאכן אז דער AI פירט זיך פראפעסיאנעל, רעספעקטפול, און לויט די פירמע'ס כללים ווען ער ענטפערט אויף קונים' פראגעס.

דער ערשטער טייל, באצייכנט ווי "instructions", זאגט קלאר פאר'ן אסיסטענט וועלכע ראלע ער שפילט, א קונה-סערוויס אסיסטענט, און באטאנט דעם טאן וואס ער מוז האלטן: אלעמאל פריינטלעך און רעספעקטפול. אזוי ענטפערט ער הילפבאר, ווי א ריכטיגער סופאָרט-אגענט.

דערנאך קומט דער "policy" אפטייל, וואס לייגט אראפ זעקס ספעציפישע געביטן וואס דער אסיסטענט מוז אנהאלטן:

- באהאנדלען צוריקצאלונגען (refunds).
- פארשרייבן קלאגעס.
- געבן געוויע אינפארמאציע ווען די פראדוקטן.
- האלטן א פראפעסיאנעלע אויפפירונג.
- פאלגן די כללים און די רעגולאציעס פון דאטא-פריוואטקייט.
- אפזאגן צו ענטפערן אויף פראגעס וואס זענען אינדרויסן פון די טעמעס.

למשל, ביי צוריקצאלונגען טאר דער אסיסטענט זיי פראצעסירן, אבער נאר לויט AnyCorp'ס כללים, און ער מוז אלעס ריכטיג דאקומענטירן. ווען א קונה קלאגט זיך, דארף דער אסיסטענט גוט צוהערן, אים בארואיגן, און ווען נויטיג אריבערגעבן דעם ענין העכער (escalate). ווען מען פרעגט אים ווען טעמעס וואס האבן נישט קיין שייכות צו די געביטן, מוז ער העפלעך אפזאגן און צוריקפירן דעם באניצער צו די ערלויבטע טעמעס.

דער לעצטער טייל איז די באניצער-פראגע (user query), דאס הייסט דער קונה'ס אמת'דיגער מעסעדזש: "הי, איך וויל צוריקגעבן דעם קאסטן וואס איך האב ביי אייך געקויפט, ווייל ער איז נישט געווען ווי באשריבן." דאס פאלט אונטער דער קאטעגאריע פון צוריקצאלונגען אין דער פאליסי. וויבאלד דער אסיסטענט טאר באהאנדלען אזא פארלאנג, ווען ער דעם מעסעדזש פראצעסירן דערלויט, מעגליך דורך באשטעטיגן דעם פראבלעם, אנהייבן דעם צוריקגעב-פראצעס, און פארשרייבן די קלאגע.

בקיצור, דער סטרוקטורירטער פראמפט שאפט א זיכערע און קלארע ראם פאר'ן אסיסטענט. ער זאגט פאר'ן AI וועלכע סארט הילף ער טאר געבן, ווי ער זאל זיך פירן, וועלכע טעמעס זענען ערלויבט, און ווי צו רעאגירן ווען ער באגעגנט א קונה'ס פראבלעם. דערביי פילט זיך דער קונה געהערט און רעספעקטירט. נאכדעם דרוקט מען אויס דעם סטרוקטורירטן פראמפט, און מען זעט אים אויפן עקראן.

שיקן דעם פראמפט צום מאדעל דורך דער OpenAI API (25:50:28)

דער נעקסטער קאוד איז א Python פראגראם וואס באנוצט די OpenAI API. די API לאזט פראגראמירער ארבעטן מיט AI מאדעלן ווי ChatGPT. די שורה דא שיקט א פראמפט צום AI און באקומט צוריק אן ענטפער. די פונקציע וואס מען רופט איז:

```
response = client.chat.completions.create(model=O1_MODEL, messages=[{"role": "user", "content": structured_prompt}])
```

עס ווערט ערקלערט יעדער טייל:

- client איז דער OpenAI API קליענט, וואס איז מסתמא אויפגעשטעלט געווארן פריער אינעם קאוד. ער פארבינדט דיין קאוד מיטן OpenAI סערווער.
- דער טייל chat.completions.create איז דער אופן ווי מען שיקט א שמועס-מעסעדזש צום AI און בעט אים פארענדיקן דעם שמועס, דאס הייסט שאפן אן ענטפער.

• דער ערשטער ארגומענט, model, זאגט פאר דער API וועלכע ווערסיע פונעם מאדעל צו באנוצן, למשל GPT-4, GPT-3.5 Turbo, אדער אן אנדער מאדעל וואס איז אויסגעשטעלט אדער fine-tuned, אויפגעהיטן אין א ווערייעבל. אזוי איז מען זיכער אז מען ניצט דעם ריכטיגן "AI-מוח" פאר דער ארבעט.

• דער צווייטער ארגומענט, messages, איז וואו מען גיט אריין די שמועס-היסטאריע אדער דעם אינפוט, פארמאטירט ווי א ליסטע פון מעסעדזש-אביעקטן. דא איז נאר איין מעסעדזש אין דער ליסטע, מיט צוויי הויפט-טיילן: role און content.

• role איז "user", דאס זאגט פאר'ן AI אז דער אינהאלט קומט פונעם באניצער, נישט פון "system" אדער "assistant".

• content פאראייניגט דעם פריערדיגן סטרוקטורירטן פראמפט, מיט די אנווייזונגען און די פאליסי-כללים, מיט דער באניצער'ס פראגע, למשל דאס צוריקגעבן דעם קאסטן. דער גאנצער צוזאמענגעשטעלטער אינהאלט איז דער מעסעדזש וואס דער AI לייענט איידער ער ענטפערט.

אין פשוט'ע ווערטער זאגט די שורה פאר'ן AI: "דו ביסט א קונה-סערוויס אגענט פאר AnyCorp. דא איז דיין פאליסי. און דא איז א קונה'ס פראגע. וואס וועט זיין דיין ענטפער?" דער AI לייענט דעם גאנצן אינפוט, פאלגט די אנווייזונגען און די כללים, און גיט צוריק א פריינטלעכן ענטפער פאר'ן קונה. דער ענטפער ווערט אויפגעהיטן אינעם ווערייעבל response, וואס מען קען שפעטער אויסדרוקן אדער ווייטער באארבעטן. מען לאזט לויפן דעם קאוד, דרוקט אויס דעם ענטפער, און מען זעט אים.

פרובירן צי דער AI קען זאגן "ניין" (25:53:01)

דערנאך קומט דער ווערייעבל refusal_input. ער אנטהאלט א באניצער-פראגע וואס בעט דעם AI: "שרייב מיר א haiku וועגן ווי גוט reasoning מאדעלן זענען." די פראגע איז איינגעוויקלט אין XML-ענליכע טעגס. דער ציל פון דעם אינפוט איז צו פרובירן דעם AI'ס פעאיקייט צו זאגן "ניין". לויטן סטרוקטורירטן פראמפט און דער פאליסי טאר דער אסיסטענט נאר רעדן וועגן צוריקצאלונגען, קלאגעס, און פראדוקט-אינפארמאציע וועגן די לאגער-לעזונגען. א haiku וועגן reasoning מאדעלן פאלט נישט אריין אין קיינע פון די טעמעס. דעריבער, וועז מען פאראייניגט דעם refusal_input מיטן סטרוקטורירטן פראמפט און שיקט עס צום AI, דארף ער העפלעך אפזאגן און ערקלערן אז ער קען נאר העלפן מיט די ערלויבטע טעמעס.

די שורה וואס וועלט געלאפן שיקט א נייעם מעסעדזש צום מאדעל, אבער דאס מאל מיט refusal_input אנשטאט user_input. ווידער רופט client.chat.completions.create די OpenAI API, model איז o1, role איז "user", און content איז דער סטרוקטורירטער פראמפט פלוס refusal_input. דאס פאראייניגט די כללים פונעם פראמפט מיט א נייער פראגע וואס איז מעגליך אינדרויסן פון די ערלויבטע טעמעס. וויבאלד דער פראמפט זאגט אז דער אסיסטענט מוז אפזאגן אויף פראגעס אויסער צוריקצאלונגען, קלאגעס און פראדוקט-אינפארמאציע, פרובירט דער טעסט צי דער AI זאגט ריכטיג אפ. בקיצור, די שורה טעסט צי דער AI פאלגט די פאליסי און זאגט העפלעך אפ אויף א פראגע וואס ער טאר נישט באהאנדלען. מען לאזט עס לויפן, און די קומענדיגע שורה ווייזט דעם AI'ס ענטפער אויפן refusal_input. מען זעט דעם אויסגאנג.

א לעגאלער אסיסטענט: base_prompt און legal_query (25:54:34)

דער נעקסטער שריט איז צו שאפן א base_prompt און א legal_query. דער ווערייעבל base_prompt באשטימט די ראלע און די אויפפירונג פונעם AI אסיסטענט. דא זאגט מען פאר'ן AI ער זאל זיך פירן ווי אן אדוואקאט וואס ספעציאליזירט זיך אין קאנקורענץ-געזעץ (competition law), און העלפט ספעציעל

געשעפטס-אייגנטימער פארשטיין די לעגאלע ענינים. דער פראמפט איז איינגעוויקלט אין טעגס ווי prompt און policy, כדי ער זאל זיין סטרוקטורירט און גרינג פאר'ן AI צו פארשטיין.

דער policy אפטייל לייגט אראפ די וויכטיגע כללים: דער אסיסטענט דארף געבן קלארע און געזויע אינפארמאציע וועגן קאנקורענץ-געזעץ, האלטן סודיות און פראפעסיאנאליטעט, און אויסמיידן צו געבן ספעציפישע לעגאלע עצות סיידן ער האט גענוג קאנטעקסט.

דער ווערייבל legal_query אנטהאלט די אמת'דיגע פראגע פונעם באניצער: "א גרעסערע פירמע באט אן אינסענטיוון פאר סאפלייערס זיי זאלן נישט טון קיין געשעפט מיט מיר. איז דאס לעגאל?" די פראגע איז איינגעוויקלט אין א query טעג, וואס מאכט דעם אינפוט סטרוקטורירט און גלייך מיטן רעשט פונעם פראמפט-פארמאט. בקיצור, די אויפשטעלונג גיט דעם AI א קלאר-באשטימטע ראלע ווי א לעגאלער מומחה, מיט א פאליסי צו פאלגן.

דערנאך שיקט די שורה די סטרוקטורירטע לעגאלע פראגע צום AI דורך דער OpenAI API. דער ציל איז צו באקומען א הילפבארן און פראפעסיאנעלן ענטפער פון א AI וואס שפילט א מומחה אין קאנקורענץ-געזעץ. ווידער ניצט מען client.chat.completions.create, role איז "user", און content איז base_prompt פלוס legal_query. מען לאזט עס לויפן.

דערנאך ווייזט מען דעם AI'ס ענטפער וויזועל, מיט סטיילינג און פארמאט, כדי דער אויסגאנג זאל אויסזען ריין און ארגאניזירט פאר'ן באניצער. מען זעט אז דער ענטפער איז א שריט-נאך-שריט אנווייזער.

א מוסטער-ביישפיל אינעם פראמפט: example_prompt (25:56:39)

דער קומענדיגער טייל איז אויך א גרויסער פראמפט. דער ווערייבל example_prompt איז א סטרוקטורירטער ביישפיל-פראמפט וואס ווייזט ווי אן AI אסיסטענט, וואס שפילט א מומחה אין קאנקורענץ-געזעץ, דארף ענטפערן אויף א באניצער'ס לעגאלער פראגע. ער איז פארמאטירט מיט ספעציעלע טעגס: prompt, policy, example, question און response, כדי אלעס זאל זיין ארגאניזירט און לייענבאר פאר א מאשין.

- אינעם prompt אפטייל באקומט דער AI די ראלע פון אן אדוואקאט וואס ספעציאליזירט זיך אין קאנקורענץ-געזעץ און העלפט געשעפטס-אייגנטימער מיט זייערע לעגאלע פראגעס. דאס שטעלט אויף די בינע ווי ער זאל ענטפערן.
- דער policy אפטייל באשטימט די גרעניצן וואס דער AI מוז אנהאלטן.
- דער example אפטייל גיט א מוסטער לעגאלע סיטואציע און ווי דער AI דארף ענטפערן.
- דער response וואס קומט נאך איז א גוט-סטרוקטורירטער, פראפעסיאנעלער ענטפער. ער הייבט אן מיט א רעפערענץ צו די אמעריקאנער אנטי-טראסט געזעצן, ספעציעל דער Sherman Antitrust Act, און ערקלערט אז געוויסע סארטן צוזאמענארבעט, ווי פרייז-פיקסירן (price fixing) אדער צעטיילן דעם מארקעט (market division), זענען אויטאמאטיש אומלעגאל. ער ווארנט אויך פון טיילן אינפארמאציע, וואס קען זיין ריזיקאליש און פירן צו פארלעצונגען פון די אנטי-טראסט געזעצן.

בקיצור, דער example_prompt דינט ווי א וועגווייזער אדער א טריינירונג-ביישפיל וואס ווייזט ווי דער AI דארף זיך פירן אין לעגאלע שמועסן. מען לאזט עס לויפן און ווייזט ווידער דעם אויסגאנג אין א פארמאטירטן סטייל.

נייער מאדול: פלאנירן מיט o1 (25:58:42)

דערנאך גייט דער קורס איבער צום נעקסטן מאדול: פלאנירן מיט o1. אין די פריערדיגע לעקציעס האט מען זיך באקענט ווי צו סטרוקטורירן פראמפטס און פירן מאדעלן ווי o1 און o1-mini צו שפילן ספעציפישע ראלעס, צי א קונה-סערוויס אגענט צי א לעגאלער ראטגעבער. אין דער לעקציע לערנט מען ווי צו פלאנירן מיט o1. דאס ברענגט א שטארקן באנוץ-פאל וואו דער AI ענטפערט נישט נאר אויף איין פראגע, נאר שאפט א פולן פלאן צו לייזן א ארבעט.

עס ווערט ערקלערט אז אזא סארט פראבלעם-לייזן פאסט ספעציעל פאר o1 מאדעלן, בפרט ווען מען ארבעט מיט א פעסטן סכום געצייג (tools) און געוויסע באגרענעצונגען. אנשטאט בעטן דעם מאדעל ער זאל אליין טון יעדן שריט, וואס קען זיין פאמעלעך און טייער, לאזט מען o1-mini שאפן דעם פלאן, און דערנאך ניצט מען א קלענערן, שנעלערן מאדעל, ווי GPT-4o mini , צו אויספירן די שריטן. די צוויי-שריטיגע מעטאד, פלאנירן מיט o1-mini און אויספירן מיט GPT-4o mini, העלפט ארבעטן עפעקטיוו.

עס ווערט געווארנט מען זאל זיך נישט זארגן אויב די רעזולטאטן שטימען נישט פונקט מיט די אין דער לעקציע. דאס איז גאנץ נארמאל, ווייל AI ענטפערס ענדערן זיך א ביסל יעדעס מאל מען לאזט זיי לויפן. בקיצור, די לעקציע בויט אויף דעם וואס מען האט שוין געלערנט און גייט א שריט ווייטער: מען לערנט דעם AI צו טראכטן פאראויס און פלאנירן, נישט נאר רעאגירן.

צוגרייטן דעם קאוד פאר'ן פלאנירן (26:00:14)

מען הייבט אן מיטן ערשטן טייל פונעם קאוד:

- דער ערשטער טייל שטילט איין די ווארענונג-מעסעדזשעס (warnings), ווי מען האט שוין פריער געזעדט. Python ווייזט אמאל ווארענונגען ווען עס באמערקט עפעס וואס מאכט נישט גלייך א טעות, אבער קען שפעטער פירן צו פראבלעמען.
- דער צווייטער טייל אימפארטירט זיכער דעם OpenAI API שליסל. אנשטאט שרייבן דעם שליסל גלייך אינעם נאטבוק, וואס איז נישט זיכער, רופט דער קאוד א הילפס-פונקציע: from helper import get_openai_api_key ברענגט אריין א פונקציע מיטן נאמען get_openai_api_key.

דערנאך:

- די ערשטע צוויי שורות אימפארטירן צוויי סטאנדארד Python מאדולן: copy און json. דער copy מאדול ווערט גענוצט ווען מען וויל דופליצירן Python אביעקטן, בפרט ווען מען דארף מאכן ענדערונגען אן אנדערן דעם ארגינעל. דער json מאדול ווערט גענוצט צו ארבעטן מיט JSON דאטא, א פארשפרייטער פארמאט צו שיקן און באקומען סטרוקטורירטע אינפארמאציע, ספעציעל ביים ארבעטן מיט APIs.
- די נעקסטע שורה אימפארטירט דעם OpenAI קלאס פונעם openai פעקל. דאס לאזט דיין קאוד רעדן מיט די OpenAI שפראך-מאדעלן ווי GPT-4 אדער o1-mini.
- דערנאך אימפארטירט מען די o1 tools פון א באזונדערן פייל מיטן נאמען utils.py, וואס אנטהאלט מסתמא געוויסע געמיינזאמע הילפס-פונקציעס אדער געצייג.
- דערנאך שטעלט מען אויף די פארבינדונג מיט דער OpenAI API דורך שאפן א client אביעקט: client = OpenAI(api_key=openai_api_key) . דער שליסל, וואס מען האט פריער באקומען מיט get_openai_api_key, ווערט אריינגעגעבן כדי צו אויטענטיפיצירן די סעסיע מיט OpenAI . מיט דעם client אביעקט וועט מען שיקן פראמפטס און באקומען ענטפערס פון די מאדעלן.

• צום סוף באשטימט מען צוויי ווערי׳עבלס וואס זאגן וועלכע מאדעלן צו באנוצן; דאס זענען פשוט נעמען פון מאדעלן.

פלאנירן און אויספירן: סופליי-טשיין און קאוד-פארגלייך (26:02:18)

צוויי מאדעלן: איינער פלאנירט, דער אנדערער פירט אויס (26:02:18)

דער קורס ערקלערט אז אין קאוד ווערן דעפינירט צוויי קורצע נעמען פאר די מאדעלן, אזוי אז מען דארף נישט יעדעס מאל אויסשרייבן דעם פולן נאמען פונעם מאדעל. מען רופט זיי פשוט "01 model" און "GPT model". דער o1-mini ווערט געוונטצט צו פלאנירן די אויפגאבע. דער GPT-4o-mini ווערט, לויט ווי עס ווערט ערקלערט, געוונטצט צו אויספירן די באזונדערע טריט פונעם פלאן.

די ערשטע שורה קאוד שאפט א ליידיגע ליסטע מיטן נאמען message list. די ליסטע ווערט היטן דעם סדר פון די מעסעדזשעס וואס גייען אהין און צוריק צווישן דעם באניצער און דעם AI-מאדעל.

דער "קאנטעקסט": א ביזנעס-סצענאריא (26:02:48)

נאכדעם ווערט דעפינירט א dictionary מיטן נאמען context. ער שטעלט אויף דעם אנהייב-מצב פון א ביזנעס-סביבה, אויף וועלכער דער מאדעל וועט פלאנירן. מען זעט די פאלגנדע טיילן:

•) Inventoryסחורה אויפן לאגער): עס ליגן איצט 50 שטיק פונעם פראדוקט Smart Home Hub X200.

•) Ordersבאשטעלונגען): עס איז דא איין באשטעלונג, ORD3001. זי בעט 200 שטיק פונעם פראדוקט פאר א קונה מיט דעם ID CUS9001, וואס געפינט זיך אין Los Angeles.

•) Suppliersליפעראנטן): עס זענען אויסגערעכנט צוויי ליפעראנטן מיט די טיילן וואס יעדער קען ליפערן. נאר SUPP1101 האט דעם טייל COMP-X200, וואס מען דארף כדי צו מאכן דעם X200, און ער האט 500 שטיק צו דער האנט.

•) Production capacityפראדוקציע-קאפאציטעט): מען קען פראדוצירן 100 שטיק גלייך, און נאך 150 די קומענדיגע וואך. דאס העלפט דעם מאדעל באשליסן ווי אזוי צו דעקן די פאדערונג פון 200 שטיק.

•) Shipping optionsשיק-מעגליכקייטן): עס זענען דא צוויי אפציעס צו ליפערן קיין Los Angeles. "Standard" קאסט $1,000 און נעמט פינף טעג. "Express" קאסט $1,500 און נעמט נאר צוויי טעג. דער מאדעל קען וועלן לויט ווי דרינגענד די באשטעלונג איז אדער לויטן בודזשעט.

•) Customersקונים): דא שטייען די פרטים פונעם קונה וואס האט באשטעלט, ווי זיין נאמען און זיין אדרעס.

•) Products פראדוקטן): דא שטייט וועלכע טיילן מען דארף צו פראדוצירן יעדן פראדוקט. פאר א פארטיגן X200 דארף מען איין שטיק COMP-X200.

דער קורס פאסט צוואם אז דער קאוד שטעלט אויף א רעאליסטישן ביזנעס-סצענאריא. ער נעמט אריין דעם פראדוקט, דעם לאגער, די באשטעלונגען, די פראדוקציע, די ליפעראנטן און דאס שיקן. דאס זענען אלע הויפט-עלעמענטן וואס דער AI דארף כדי צו פלאנירן ווי אזוי צו דורכפירן די באשטעלונג אויף א גוטן אופן.

דער פראמפט פאר פלאנירן (26:04:22)

דער נעקסטער בלאק דעפינירט א לאנגן string מיטן נאמען o1 prompt. דאס איז א פראמפט מיט אנווייזונגען פאר א פארגעשריטענעם שפראך-מאדעל ווי o1. דער ציל פונעם פראמפט איז צו פירן דעם מאדעל, אז ער זאל

זיך פירן ווי א געהילף פאר סופליי-טשעין פארוואלטונג. ער זאל אויסארבעטן א פרטימדיגן, לאגישן פלאן ווי אזוי צו באהאנדלען א קאמפליצירטע אויפגאבע פון דורכפירן א באשטעלונג. לויטן קורס איז דאס "דער מוח", וואס צעטיילט א ביזנעס-פראבלעם און צייכנט אן די בעסטע טריט ווי אזוי עס צו לייזן.

- דער ערשטער פאראגראף שטעלט אויף דעם קאנטעקסט. ער זאגט דעם AI אז ער וועט באקומען א קאמפליצירטע אויפגאבע. די אויפגאבע נעמט איריין פארוואלטן קונים, דעם לאגער, פראדוקציע, ליפערונגען און שיקן.
- דער נעקסטער טייל רעכנט אויס אלע פונקציעס וואס דער אויספירער-AI האט צו דער האנט. די פונקציעס לאזן אים טון זאכן ווי: קאנטראלירן דעם לאגער, צוטיילן סחורה פאר באשטעלונגען, אריינגעבן באשטעלונגען ביי ליפעראנטן, איינשטעלן פראדוקציע-ראנדן, און אזוי ווייטער. ביי יעדע פונקציע איז דא א קורצע באשרייבונג, צוזאמען מיט וואס זי דארף באקומען און וואס מען ערווארט פון איר.

דאס זארגט אז דער פלאנירנדיגער AI זאל וויסן וועלכע כלים עס זענען דא, ווי זיי ארבעטן און וואס יעדער איז געמיינט צו טון. דערמיט קען דער מאדעל בויען א פלאן וואס איז רעאליסטיש, און וואס דער אגענט וואס קומט נאכדעם קען טאקע אויספירן.

אין קורצן: דער o1 prompt שטעלט אויף א רייכע, גוט-אויסגעשטעלטע סביבה פון אנווייזונגען פארן פלאנירנדיגן מאדעל. ער הייסט דעם AI טראכטן גוט, טראכטן טריט נאך טריט, און שאפן א קלארן, פרטימדיגן און לאגישן פלאן פאר דער באשטעלונג. דער פלאן דארף זיין אזוי, אז אן אנדער AI זאל אים קענען זיכער אויספירן מיט די כלים וואס מען האט געגעבן. דאס זארגט פאר עקביות, זיכערקייט און קלארקייט אין יעדן טריט פונעם סופליי-טשעין פראצעס.

דער סיסטעם-פראמפט פארן אויספירער (26:06:24)

דער נעקסטער בלאק דעפינירט א משתנה מיטן נאמען GPT4 system prompt. דאס איז א string וואס פירט דאס פירן זיך פונעם אויספירנדיגן מאדעל, דאס הייסט דעם GPT-4o-mini. דער o1-mini איז פאראנטווארטליך פארן פלאנירן. דער פראמפט איז פארן מאדעל וואס פירט אויס יעדן טריט פונעם פלאן. מען זאגט דעם געהילף אז ער זאל זיין א הילפסבארער אויספירער. זיין ארבעט איז צו פאלגן געוי א געגעבענע פאליסי ווי אזוי צו באהאנדלען באשטעלונגען וואס קומען אריין.

דער פראמפט שטעלט אויף א סדר פון טריט:

- ערשט זאל דער מאדעל לייענען און פארשטיין די גאנצע פאליסי.
- נאכדעם זאל ער אידענטיפיצירן אויף וועלכן טריט פון דער פאליסי ער ארבעט יעצט.
- איידער ער טוט עפעס, זאל ער געבן א קורצע ערקלערונג פאר זיין באשלוס.
- צום סוף זאל ער אויספירן די אקציע. געווענליך מיינט דאס רופן איינע פון די פונקציעס מיט די ריכטיגע אינפוטס.

אין קורצן: דער סיסטעם-פראמפט שטעלט אויף ווי דער אויספירער טראכט און ווי ער פירט זיך. דער מאדעל ארבעט טריט נאך טריט, ער ערקלערט זיין געדאנקען-גאנג, און ער טוט נאר אקציעס וואס פאסן מיט דער פאליסי. דאס שאפט א קאנטראלירטן און דורכזיכטיגן אויספיר-פלוס פאר באהאנדלען באשטעלונגען אין א סופליי-טשעין סצענאריא.

די tools-ליסטע: דער "געצייג-קאסטן" (26:07:55)

דער נעקסטער בלאק דעפינירט א ליסטע מיטן נאמען tools. זי האלט א זאמלונג פונקציעס וואס דער AI קען נוצן צו ארבעטן מיטן סופליי-טשעין סיסטעם. די פונקציעס זענען ווי א געצייג-קאסטן. יעדע פונקציע טוט א באשטימטע ארבעט וואס האט צו טון מיטן לאגער, פראדוקציע, ליפעראנטן, באשטעלונגען, א.א.וו.

יעדער tool איז דעפינירט אלס dictionary. ער האט א type, וועלכער איז function, און א function-טייל. דער function-טייל גיט דעם נאמען פון דער פונקציע, א באשרייבונג וואס זי טוט, און די פאראמעטערס וואס זי ערווארט. די הויפט-פונקציעס זענען:

- get_inventory_status: קאנטראלירט וויפיל פון א פראדוקט ליגט אויפן לאגער.
- update_inventory: לייגט צו אדער נעמט אראפ סחורה פונעם לאגער.
- fetch_new_orders: ווייזט וועלכע באשטעלונגען ווארטן נאך.
- get_product_details און check_available_suppliers: געפינען אויס וואס מען דארף צו מאכן א פראדוקט, אדער געפינען ליפעראנטן.
- place_purchase_order: באשטעלט טיילן.
- allocate_stock: רעזערווירט סחורה פאר א קונה'ס באשטעלונג.
- check_production_capacity און schedule_production_run: פלאנירן און פירן אויס די פראדוקציע.
- calculate_shipping_options און book_shipment: באהאנדלען די ליפערונג.
- send_order_update: גיט צו וויסן דעם קונה.
- instructions_complete: צייכנט אן אז אלע טריט פונעם פלאן זענען פארטיג.

יעדע פונקציע דעפינירט אויך די פאראמעטערס וואס זי פאדערט, ווי product ID, order ID, quantity, א.א.וו. דער קורס באטאנט אז דאס איז וויכטיג, ווייל דער AI מוז געבן פונקט די ריכטיגע דאטא ווען ער רופט די כלים. דערפאר נוצט מען required-פעלדער און דעם סעטינג additional properties . אין קורצן: די tools-ליסטע דעפינירט א פולן סעט פונקציעס, מיט ווען דער AI-אגענט קען פירן דעם סופליי-טשעין סיסטעם.

די Python-פונקציעס וואס סימולירן די אפעראציעס (26:09:25)

דער בלאק דעפינירט אלע Python-פונקציעס וואס סימולירן די סופליי-טשעין אפעראציעס וואס מען האט פריער באשריבן.

- לאגער-פארוואלטונג: get_product_details ברענגט די אינפארמאציע וועגן א פראדוקט. get_inventory_status גיט צוריק וויפיל שטיק פון א געוויסן פראדוקט ליגן איצט אויפן לאגער. update_inventory(product_id, quantity_change) פאסט צו די צאל אויפן לאגער לויט א פאזיטיווער אדער נעגאטיווער ענדערונג. זי זארגט אז דער לאגער זאל קיינמאל נישט פאלן אונטער נול.
- allocate_stock פרובירט צו רעזערווירן סחורה פאר א באשטעלונג. אויב עס איז דא גענוג סחורה, פארמינערט זי דעם לאגער און באשטעטיגט די רעזערוואציע. אויב נישט, טיילט זי צו וואס עס איז יא דא, און צייכנט אן דעם רעזולטאט מיט א ווארענונג אז עס איז נישט גענוג סחורה.

• ליפעראנטן: check_available_suppliers גיט צוריק די ליסטע ליפעראנטן פון וועמען די פירמע קען באשטעלן טיילן.

• פראדוקציע: check_production_capacity(time_frame) קאנטראלירט וויפיל שטיק מען קען פראדוצירן אין דעם געגעבענעם צייט-ראם. schedule_production_run קאנטראלירט די קאפאציטעט, שטעלט איין די פראדוקציע אויב עס איז מעגליך, און דערהיינטיגט דעם לאגער.

• שיקן: calculate_shipping_options גיט צוריק א ליסטע פון די שיק-מעטאדן וואס זענען דא, מיט זייערע פרייזן פאר דעם באשטימטן ציל-ארט.

• צום לעצט, send_order_update סימולירט אז מען שיקט א מעסעדזש צום קונה, כדי ער זאל וויסן וואס טוט זיך מיט זיין באשטעלונג.

גאנץ צום סוף איז דא א dictionary מיטן נאמען function_mapping. ער פארבינדט יעדן פונקציע-נאמען מיט דער פאקטישער Python-פונקציע. מען לויפט דעם בלאק, און דער קורס מוטיגט צו פארבינדן צוזאמען דעם גאנצן פראצעס און ארויסנעמען דערפון דאס מערסטע.

א נייער מאדול: קאודירן מיט o1 – אויפשטעלן די סביבה (26:11:28)

א נייער מאדול הייבט זיך אן וועגן קאודירן מיט o1 . צום ערשט באהאלט מען די ווארענונגען מיט)"ignore"warnings.filterwarnings(, אזוי ווי מען האט שוין פריער געטון. דערמיט בלייבט דער אויטפוט ריין, ווייל ווארענונגען וואס זענען נישט קריטיש ווייזן זיך נישט ווען דער קאוד לויפט. נאכדעם אימפארטירט מען די פונקציע get_openai_api_key פונעם helper-טעקע. מען רופט די פונקציע, און דעם רעזולטאט היט מען אין דעם משתנה openai_api_key. אין קורצן: די אויפשטעלונג באהאלט ווארענונגען און לאדנט אויף א זיכערן אופן דעם API-שליסל, כדי צו נוצן די OpenAI כלים.

נאכדעם אימפארטירט מען דריי ניצליכע כלים פון IPython.display: display, Image און Markdown. מען נוצט זיי אפט אין Jupyter notebooks, כדי צו וויזן בילדער, פארמאטירן אויטפוט מיט markdown, אדער וויזן אביעקטן אויף א ריינערן אופן. מען אימפארטירט אויך דעם קלאס OpenAI פון דער OpenAI ביבליאטעק. דער קלאס שאפט א קליענט וואס רעדט מיטן API פון OpenAI, כדי צו באקומען ענטפערס, לויפן מאדעלן, אדער טון אנדערע AI-ארבעטן. די שורה

OpenAI(api_key=openai_api_key) = client

שטעלט אויף דעם קליענט מיטן שליסל וואס מען האט פריער געהיטן. דאס פארבינדט דעם קאוד מיט די OpenAI סערוויסעס, אזוי אז מען קען אנהייבן מאכן API -רופן. צום סוף דעפינירט מען צוויי משתנים: GPT_MODEL מיט "gpt-4o-mini", און O1_MODEL מיט "o1-mini". דאס זענען קורצע צייכנס, מיט וועלכע מען באשטימט אין קאוד מיט וועלכן מאדעל מען וויל ארבעטן ביי יעדן API-רוף.

די פונקציע get_chat_completion (26:13:00)

דער קאוד דעפינירט א פונקציע מיטן נאמען get_chat_completion. זי ברענגט א ענטפער פון א טשאט-מאדעל פון OpenAI. זי נעמט צוויי אינפוטס: דעם נאמען פונעם מאדעל, למשל gpt-4o-mini, און דעם באניצער'ס מעסעדזש אדער פראגע. אינעווייניג רופט זי client.chat.completions.create. דאס איז די מעטאד אין OpenAI'ס API צו שיקן דעם פראמפט צום אויסגעקליבענעם מאדעל און באקומען א טשאט-ענטפער. די מעסעדזשעס גייען אריבער אין דעם פארמאט וואס OpenAI ערווארט: א ליסטע פון מעסעדזש-אביעקטן, יעדער מיט א ראלע, ווי user אדער assistant. צום סוף גיט די פונקציע צוריק נאר דעם

אינהאלט פונעם מאדעל'ס ענטפער, דורך response.choices[0].message.content. אזוי בלייבט דער אויטפוט ריין: מען באקומט נאר דעם טעקסט פונעם AI'ס ענטפער, און נישט די גאנצע סטרוקטור פונעם API-ענטפער.

א פראמפט צו בויען א React-קאמפאנענט (26:14:01)

דער קאוד דעפינירט א string אויף עטליכע שורות, מיטן נאמען react_demo_prompt. דאס איז א פראמפט געשריבן אין נאטירלעכער שפראך, וואס מען שיקט צו א AI-מאדעל ווי ChatGPT. דער פראמפט בעט דעם AI צו שאפן א React-קאמפאנענט מיטן נאמען FeedbackForm, צו זאמלען סטרוקטורירטע פידבעק נאך אן אינטערוויו. ער שטעלט פיר הויפט-פאדערונגען:

- אפשאצן קאנדידאטן לויט) rubricsקריטעריעס).
- געבן קאנקרעטע באווייזן פאר יעדער אפשאצונג.
- אויטאמאטיש אויסרעכענען א רעקאמענדאציע-ציפער.
- פירן דעם אינטערוויוער צו געבן פידבעק וועגן דעם קאנדידאט'ס אויפפירונג.

דער פראמפט האט אויך נאך אנווייזונגען: נוצן קלוגע UX-אייגנשאפטן, ווי גוט-דורכגעטראכטע rubrics, קלוגע וואליդאציע און הילפסבארע אנווייזונגען. ער האט אויך קלארע קאודירונג-אנווייזונגען, למשל אז דער קאמפאנענט מוז זיך אנהייבן מיט "use client". אין קורצן: דער פראמפט איז געמאכט צו שאפן א פיינעם, קלוגן פארם פאר אינטערוויו-פידבעק אין React.

פארגלייכן GPT-4o-mini מיט o1-mini (26:15:01)

מען רופט get_chat_completion מיט צוויי ארגומענטן: GPT_MODEL, וואס באשטימט דעם מאדעל (gpt-4o-mini), און react_demo_prompt, וואס איז די פרטימדיגע אנווייזונג צו שאפן דעם React-קאמפאנענט. די פונקציע שיקט דעם פראמפט צום מאדעל און גיט צוריק דעם ענטפער, וואס איז א בלאק קאוד אזוי ווי מען האט געבעטן. מען היט דעם רעזולטאט אינעם משתנה gpt_code. מען לויפט דאס און מען דרוקט אויס דעם קאוד, און מען זעט דעם קאוד אויפן עקראן.

נאכדעם ווייזט מען א בילד מיט)"gpt4_app_image.png"display(Image(). Image לאדנט די טעקע gpt4_app_image.png פונעם יעצטיגן דירעקטאריום, און display ווייזט זי אינעם אויטפוט פונעם notebook. דאס איז ניצליך צו ווייזן וויזועלע רעזולטאטן, בילדער פון דער UI, אדער דיאגראמען לעבן דעם קאוד.

דערנאך שיקט מען דעם זעלבן react_demo_prompt צו אן אנדער מאדעל, O1_MODEL, דאס הייסט o1-mini, ווידער דורך get_chat_completion. דער רעזולטאט ווערט געהיטן אין o1_code. דאס לאזט פארגלייכן די רעזולטאטן פון די צוויי מאדעלן אויף דעם זעלבן פראמפט. מען דרוקט אויס דעם o1-קאוד, מען זעט אז ער איז אויסגעדרוקט, און מען קען אים יעצט אויך ווייזן.

קאוד-רעוויו און אנאליזירן בילדער מיט o1 (26:17:22)

פארבעסערן א Python פונקציע מיט צוויי מאדעלן (26:17:22)

דער קורס האלט ביי א ביישפיל פון א פונקציע וואס הייסט process_orders. זי נעמט א רשימה פון באשטעלונגען (orders) און צוויי אפציאנעלע פאראמעטערס: settings, debug און notify_customer. די

פונקציע באהאנדלט יעדע באשטעלונג: זי פרובירט אויס (validate) אירע פעלדער, רעכנט אויס די סכומים, לייגט צו הנחות (discounts), און שטעלט אין ריי א מעלדונג צום קונה. די רעזולטאטן, די טעותים און די מעלדונגען זאמלט זי יעדעס אין א באזונדערער רשימה, און צום סוף גיט זי זיי צוריק אין א dictionary.

עס ווערט אבער אנגעוויזן אויף עטליכע פראבלעמען אינעם קאוד:

- א "mutable default argument" פאר settings איז א ריזיקע, ווייל עס קען ברענגען אומגעריכטע רעזולטאטן. בעסער צו שטעלן None אלס דיפאלט, און ערשט אינעווייניג אין דער פונקציע צוטיילן א ליידיגן dictionary.
- א פארגלייך ווי if debug == True דארף מען פארפשוטערן צו if debug.
- דאס זעלבע איז מיט if items_valid == True, וואס מען קען שרייבן פשוט if items_valid.

דער קורס פאסט צוזאם: די לאגיק פון דער פונקציע ארבעט פאר א פשוטער באהאנדלונג פון באשטעלונגען, אבער מען דארף זי אויסרייניגן לויט די "best practices" פון Python, אסאך פאר געשווינדיגקייט און פאר גרינגער אויפהאלטונג (maintainability).

דערנאך לויפט מען דעם קאוד. ער שטעלט צוזאם א פראמפט-סטרינג מיטן נאמען prompt, דורך אן f-string. דער פראמפט הייסט דעם AI אויסרייניגן און פארבעסערן דעם Python קאוד וואס שטייט אינעם ווריאבל code_snippet, און ער בעט ספעציעל אז דער מאדעל זאל צוריקגעבן נאר דעם פארבעסערטן קאוד.

- די פונקציע get_chat_completion, וואס מען האט שוין פריער דעפינירט, שיקט דעם פראמפט צום אויסגעקליבענעם GPT מאדעל. זי רופט אן דעם OpenAI API און באקומט צוריק א נייע ווערסיע פונעם קאוד.
- print(gpt_code) ווייזט די נייע ווערסיע, אז מען זאל זי קענען איבערקוקן אדער באנוצן.
- דערנאך שיקט מען דעם זעלבן פראמפט צו אן אנדער מאדעל, דעם o1 מאדעל, אויך דורך get_chat_completion. דער ענטפער ווערט אפגעהיטן אינעם ווריאבל o1_code, און ער דארף אנטהאלטן דעם אפטימירטן קאוד וואס o1 האט צוגעגרייט.
- print(o1_code) ווייזט דעם פארבעסערטן קאוד, אז מען זאל קענען פארגלייכן די ווערסיע פון o1 מיט דער פון GPT. דאס איז נוצליך כדי אפצושאצן די אונטערשיידן צווישן די מאדעלן אין דער קוואליטעט, קלארקייט און אפטימיזאציע פונעם קאוד.
- צום סוף ווערט א רעזולטאט אפגעהיטן אינעם ווריאבל result, און מען ווייזט אים מיט display(Markdown(result)). דאס שטעלט אוועק דעם ענטפער אין א שיין פארמאטירטן Markdown, וואס מאכט אים גרינג צו לייענען אין Jupyter notebooks.

דער קורס פאסט צוזאם: דער שריט ווייזט ווי מען קען באנוצן AI נישט נאר כדי צו שרייבן קאוד, נאר אויך כדי צו מאכן א שנעלן קאוד-רעוויו און אנאליז, מיט די צוויי מאדעלן זייט ביי זייט.

לעצטער מאדול: טראכטן מיט בילדער, צוגרייטונג (26:20:25)

דער לעצטער מאדול הייבט אן מיט "reasoning with images", דאס הייסט אז דער מאדעל זאל טראכטן און אויספירן לויט א בילד.

- ערשט ווערט אימפארטירט דער אינגעבויטער מאדול warnings, וואס פירט אן מיט ווארענונג-מעלדונגען בשעת דער קאוד לויפט.)"ignore"warnings.filterwarnings(באהאלט אלע ווארענונגען, אזוי אז

מעלדונגען וואס זענען נישט קריטיש ווייזן זיך נישט אינעם אויסגאנג. מען טוט דאס געווענליך כדי דער אויסגאנג זאל בלייבן ריין, בפרט אין notebooks אדער סקריפטס וואו מען וויל נישט אז ווארענונגען זאלן פארפאטשקען די רעזולטאטן.

• דערנאך אימפארטירט מען דעם מאדול json, כדי צו ארבעטן מיט JSON דאטא. ער העלפט ביים צעלייגן (parsing), פארמאטירן אדער אפהיטן אינפארמאציע וואס איז געבויט מיט א סדר.

• מען אימפארטירט די קלאס OpenAI פונעם OpenAI פעקל. דורך איר פארבינדט מען זיך מיטן OpenAI API און מען שיקט בקשות צו די מאדעלן, כדי זיי זאלן צוגרייטן טעקסט, קאוד אדער אנדערע אויסגאנגען.

• די שורה from IPython.display import display, Markdown, Image ברענגט מכשירים כדי צו ווייזן Markdown טעקסט, בילדער אדער פארמאטירטן אויסגאנג אויפן עקראן. מען באנוצט זיי געווענליך אין Jupyter notebook.

• די פונקציע get_openai_api_key ווערט אימפארטירט פון א באזונדערער טעקע וואס הייסט helper. זי ברענגט דעם OpenAI API שליסל אויף א זיכערן אופן, אזוי אז מען דארף אים נישט שרייבן גלייך אינעם קאוד (hardcoded). מען רופט זי אן, און דער שליסל ווערט אפגעהיטן אינעם ווריאבל openai_api_key. דער שליסל איז נויטיג כדי זיך צו אידענטיפיצירן ביים OpenAI API.

• מען דעפינירט די נעמען פון די מאדעלן: GPT_MODEL פאר GPT-4o mini, און O1_MODEL פאר o1. די נעמען גיט מען שפעטער אריין ווען מען רופט אן דעם API.

• client = OpenAI() שאפט א קליענט, דורך וועלכן מען שיקט פראמפטס און מען באקומט ענטפערס. עס ווערט באמערקט אז דא דארף מען אריינגעבן api_key=openai_api_key פאר דער אידענטיפיקאציע, סיידן די סביבה באהאנדלט דעם שליסל אויטאמאטיש.

ווייזן דעם בילד און עס שיקן צום מאדעל (26:22:27)

דער וועג פונעם בילד ווערט אפגעהיטן אינעם ווריאבל image_filepath. דער וועג ווייזט אז דאס בילד דארף ליגן אין א טעקע-פאלדער מיטן נאמען data אינעם פראיעקט. דער בילד איז א שעמע פון אן ארגאניזאציע (organization chart). display(Image(image_filepath)) לאדט דאס בילד און ווייזט עס גלייך אינעם אויסגאנג פונעם notebook. מען באנוצט דאס געווענליך וועג מען ארבעט מיט ויזועלער דאטא, אדער וועג מען שטעלט פאר דיאגראמען און טשארטס.

דערנאך אימפארטירט מען דעם מאדול base64. ער פארוואנדלט בינארישע דאטא, ווי למשל בילד-טעקעס, אין א base64 סטרינג. דאס איז נויטיג ווייל APIs ווי די vision מאדעלן פון OpenAI נעמען אן בילדער נאר אלס base64 סטרינגס.

• די פונקציע encode_image(image_path) עפנט די טעקע אין בינארישן מאדוס ("rb "), פארוואנדלט זי אין base64, און גיט צוריק א UTF-8 סטרינג. אזוי איז דאס בילד גרייט צו ווערן געשיקט צום מאדעל אינעם פארלאנגטן פארמאט.

• די הויפט-פונקציע o1_vision שיקט צוזאמען א בילד און א טעקסט-פראמפט צו א מאדעל פון OpenAI וואס קען זען בילדער, ווי GPT-4. קודם פארוואנדלט זי דאס בילד דורך encode_image.

• אינעווייניג קוקט זי צי json_mode איז True אדער False. אויב True, בעט מען דעם ענטפער אין א JSON פארמאט מיט סדר, דורך response_format={"type": "json_object"}. דאס איז נוצליך כדי ארויסצוציען

דאטא מיט סדר פונעם ענטפער. אויב False, שיקט מען פשוט דעם פראמפט מיטן בילד און מען ערווארט א געווענליכן טעקסט-ענטפער.

- ביידע ווערסיעס רופן אן client.chat.completions.create.
- די פונקציע גיט צוריק דעם רויען ענטפער-אביעקט פונעם API, אזוי אז דער וואס רופט זי אן קען אים ווייטער באהאנדלען אדער ווייזן.

בקיצור: די פונקציע לאזט שיקן א בילד מיט א פראמפט צו א מאדעל וואס קען זען, מיט א ברירה צו באקומען דעם ענטפער אלס פשוטן טעקסט אדער אלס JSON מיט סדר.

דערנאך רופט מען אן o1_vision מיט דריי ארגומענטן:

- file_path=image_filepath, דער וועג צום בילד.
- prompt="What is this?", וואס בעט דעם מאדעל צו באשרייבן דאס בילד.
- model=O1_MODEL, דער מאדעל וואס קען זען בילדער.

דער ענטפער ווערט אפגעהיטן אינעם ווריאבל response, און ער אנטהאלט ווי דער AI פארשטייט דאס בילד.

מען ווייזט דעם ענטפער מיט display(Markdown(response.choices[0].message.content)) . דער טייל response.choices[0].message.content ברענגט דעם טעקסט וואס דער מאדעל האט צוגעגרייט אין דער ערשטער ענטפער-ברירה. וועז מען ווילקלט אים אריין אין Markdown, ווייזט ער זיך מיט דער ריכטיגער פארמאטירונג, ווי דיקע אותיות, פונקטן אדער קעפלעך, אויב דער ענטפער האט Markdown סינטאקס. display שטעלט דאס אוועק אינעם notebook שיין צו לייענען און מיט א קלארן סדר.

ארויסציען דעם סדר פון דער ארגאניזאציע אלס JSON (26:26:02)

מען דעפינירט א ווריאבל structured_prompt. דאס איז א סטרינג וואס מען וועט שיקן צום מאדעל וואס קען זען בילדער, כדי ארויסצוציען דאטא מיט סדר פונעם בילד. דער פראמפט נוצט ספעציעלע אנווייזונג-טאגס כדי צו זאגן קלאר וואס דער מאדעל זאל טון. ער זאגט אז דער מאדעל איז א "consulting assistant", ווענס ארבעט איז צו באהאנדלען די דאטא פון דער ארגאניזאציע-שעמע אינעם בילד. דערנאך באשרייבט ער דעם געוואונטשענעם אויסגאנג: א JSON סטרוקטור וואס האט פאר יעדן מענטש:

- אן ID לויט דער ברירה (arbitrary)
- א נאמען
- א ראלע
- א רשימה (array) פון די IDs צו וועמען דער מענטש מעלדט זיך
- א רשימה פון די IDs וואס מעלדן זיך צו אים

דאס איז א ריינער און סדר'דיגער אופן צו שילדערן ווער עס שטייט איבער וועמען אין אן ארגאניזאציע. print(structured_prompt) ווייזט דעם גאנצן פראמפט, כדי מען זאל אים קענען איבערקוקן פאר מען שיקט אים צו א פונקציע ווי o1_vision.

דערנאך רופט מען אן o1_vision מיט:

- file_path=image_filepath, דער ארגאניזאציע-שעמע

- model=O1_MODEL
- prompt=structured_prompt, מיט די פרטים איבערן אויסגאנג-פארמאט
- json_mode=True, וואס זאגט דעם מאדעל צוריקצוגעבן א JSON מיט סדר אנשטאט פשוטן טעקסט. אזוי איז גרינגער עס צעצולייגן און באנוצן אין א פראגראם.

מען דרוקט אויס דעם JSON ענטפער פון דער ערשטער ברירה, און מען זעט אינעם אויסגאנג די ארגאניזאציע מיט IDs, נעמען, ראלעס און ווער מעלדט זיך צו וועמען.

דערנאך שאפט מען א וועריאבל cleaned_json מיט אן אויסגעריניגטער וועדסיע פונעם ענטפער. עס וועדט ערקלערט אז מאדעלן וויקלען אמאל איין דעם JSON אין Markdown, מיט דריי "backticks " און דעם וואדט json, כדי צו צייכענען א קאוד-בלאק. די שורה נעמט אראפ די ראמען דורך .) replaceמען פארביטן זיי מיט ליידיגע ציטירצייכנס), און עס בלייבט נאר דער רויער JSON. דער רעזולטאט איז א פשוטער JSON סטרינג וואס מען קען צעלייגן און וויטער באהאנדלען אן פראבלעמען מיט פארמאטירונג.

די קומענדיגע שורה פרובירט צעלייגן דעם ענטפער גלייך מיט json.loads. די פונקציע פארוואנדלט א JSON סטרינג אין א Python dictionary אדער list. זי נעמט o1_response.choices[0].message.content, וואס דארף אנטהאלטן דעם JSON פון דער ארגאניזאציע-שעמע, און היט אפ דעם רעזולטאט אין org_data. אזוי קען מען ארבעטן מיט דער ארגאניזאציע אלס Python אביעקט מיט סדר. עס וועדט אבער געווארנט: אויב דער ענטפער אנטהאלט נאך די Markdown ראמען, דארף מען באנוצן די פריערדיגע אויסגעריניגטע וועדסיע ביים צעלייגן.

פרעגן שאלות איבער דער ארגאניזאציע (26:29:35)

מען דעפינירט א נייעם פראמפט, analysis_prompt, וואס זאל פירן דעם מאדעל צו ענטפערן שאלות לויט דער ארגאניזאציע-שעמע וואס מען האט פריער ארויסגעצויגן. ער הייבט אן מיט ספעציעלע אנווייזונגען: דו ביסט אן עקספערט-געהילף פאר ארגאניזאציע-שעמעס, און דיין ראלע איז צו ענטפערן אויף יעדע שאלה איבער דער שעמע לויט דער ארגאניזאציע-דאטא. דאס זאגט דעם AI וועלכע ראלע ער שפילט און וויי ער זאל באנוצן די דאטא. דערנאך לייגט מען אריין די דאטא פון דער שעמע צווישן organization_data טאגס, דורך אן f-string. אזוי פארשטייט דער מאדעל קלאר אז דאס איז די דאטא וואס ער דארף נוצן ביים ענטפערן.

אינעם קאוד אימפארטירט מען נאכאמאל get_openai_api_key, כדי צו ברענגען דעם שליסל אויף א זיכערן אופן. מען רופט זי אן און היט אפ דעם שליסל אין openai_api_key, אז מען זאל אים קענען באנוצן איבעראל אינעם קאוד. client = OpenAI(...) שאפט א קליענט מיט דעם שליסל, וואס לאזט שיקן פראמפטס דורך באשטעטיגטע API רופן.

די messages האבן א role פון "user", און דער content איז analysis_prompt מיט צוויי שאלות: ווער האט די העכסטע ראנג, און וועלכער מענעדזשער האט די מערסטע מענטשן וואס מעלדן זיך צו אים. מען לויפט דאס, און מען וועיזט דעם ענטפער מיט Markdown, אזוי אז דיקע אותיות, פונקטן אדער קעפלעך וועיזן זיך ריכטיג.

א דיאגראם פון א דאטא-באזע און סוף פונעם קורס (26:31:12)

צום לעצט שטעלט מען image_filepath צו א בילד מיטן נאמען er_relationship.png אינעם פאלדער data. דאס בילד אנטהאלט א "entity relationship diagram", א שעמע פון טאבעלעס אין א דאטא-באזע און וויי זיי זענען פארבונדן. מען וועיזט עס אין Jupyter notebook מיט display(Image(...)). אינעם אויסגאנג זעט מען

פעלדער ווי: product ID, order, product, דאס ID פונעם קליענט (C_ID), product name, client address, product price און client credit card.

דערמיט ענדיגט זיך דער טייל פונעם קורס וואס הייסט "reasoning with o1".