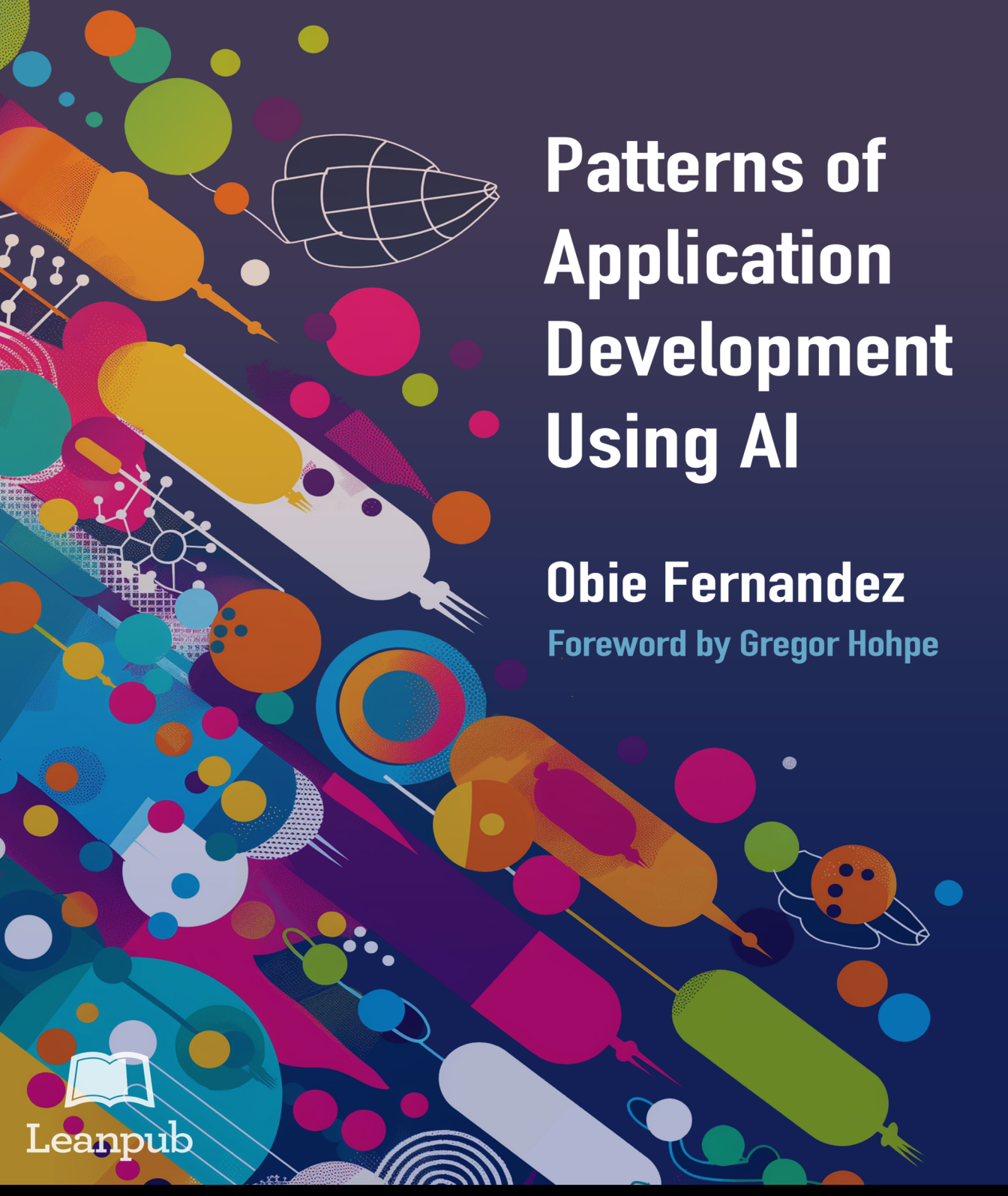


Izdanje na Srpskom

Obrasci razvoja aplikacija korišćenjem AI (Izdanje na Srpskom)

Obie Fernandez

Ova knjiga je na prodaju na
http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn

Ova verzija je objavljena 2025-01-23

Tweetujte ovu knjigu!

Molimo pomozi Obie Fernandez tako što ćeš prosiriti reč o ovoj knjizi na Twitter!

Predloženi hešteg za ovu knjigu je #poaduai.

Saznajte šta drugi ljudi kažu o knjizi klikom na ovaj link da tražite ovaj hešteg na Twitter-u:

#poaduai

Mojoj neustrašivoj kraljici, mojoj muzi, mojoj svetlosti i ljubavi, Victoriji

Also By Obie Fernandez

Patterns of Application Development Using AI

The Rails 8 Way

The Rails 7 Way

XML The Rails Way

Serverless

El Libro Principiante de Node

The Lean Enterprise

Садржај

Predgovor Gregora Hohpea . . . . . **i**

Predgovor . . . . . **ii**

O knjizi iii

O primerima koda iii

Šta ne obrađujem iii

Kome je namenjena ova knjiga iii

Izgradnja zajedničkog rečnika iii

Uključivanje iii

Zahvalnice iii

Šta je sa ilustracijama? iv

O Lean Publishing-u iv

O Autoru v

Uvod **1**

Razmišljanja o softverskoj arhitekturi 2

Šta je Veliki jezički model? 3

Razumevanje zaključivanja 5

Razmišljanje o performansama 25

Eksperimentisanje sa različitim LLM modelima 27

Složeni AI sistemi 27

Deo 1: Osnovni pristupi i tehnike 35

Suziti Put **36**

Latentni prostor: Nezamislivo ogroman 38

Kako se Put “Sužava” 42

Osnovni modeli naspram modela obučenih na instrukcijama 45

Inženjerstvo upita 52

Destilacija promptova 67

Šta je sa finim podešavanjem? 74

Generisanje potpomognuto pretraživanjem (RAG) **75**

Šta je Generisanje potpomognuto pretraživanjem? 75

Kako RAG funkcioniše? 75

Zašto koristiti RAG u vašim aplikacijama? 75

Implementacija RAG-a u Vašoj Aplikaciji 75

Segmentacija propozicija 76

Primeri RAG-a iz stvarnog sveta 76

Inteligentna Optimizacija Upita (IQO) 77

Ponovno Rangiranje 77

RAG Procena (RAGAs) 77

Izazovi i budući izgledi 79

Mnoštvo radnika **81**

AI radnici kao nezavisne komponente za višekratnu upotrebu 82

Upravljanje nalozima 84

Primene u E-trgovini 85

Primene u zdravstvu 93

AI radnik kao menadžer procesa 96

Integracija AI Radnika U Arhitekturu Vaše Aplikacije 100

Kompozabilnost i orkestracija AI radnika 103

Kombinovanje tradicionalne OPJ sa VJM-ovima 112

Upotreba alata 115

Šta je upotreba alata? 115

Potencijal upotrebe alata 117

Tok rada sa alatima 118

Najbolje prakse za upotrebu alata 131

Komponovanje i ulančavanje alata 135

Budući pravci 137

Obrada toka podataka 139

Implementacija ReplyStream-a 140

“Konverzacijska petlja” 146

Automatski Nastavak 148

Zaključak 150

Samozalečujući podaci 152

Praktična studija slučaja: Popravljanje neispravnog JSON-a 154

Razmatranja i kontraindikacije 159

Kontekstualno generisanje sadržaja 173

Personalizacija 174

Produktivnost 176

Brza iteracija i eksperimentisanje 178

AI podržana lokalizacija 180

Važnost korisničkog testiranja i povratnih informacija 182

Generativni UI 184

Generisanje teksta za korisničke interfejse 185

Definisanje generativnog UI-ja 194

Primer 196

Prelazak na dizajn orijentisan ka ishodima 198
Izazovi i razmatranja 200
Budući izgledi i mogućnosti 201

Inteligentna orkestracija radnih tokova 205
Poslovna potreba 206
Ključne prednosti 207
Ključni obrasci 207
Rukovanje Izuzecima i Oporavak 210
Implementacija orkestracije inteligentnog radnog toka u praksi 213
Praćenje i beleženje 227
Razmatranja skalabilnosti i performansi 231
Testiranje i validacija radnih tokova 236

Deo 2: Obrasci 245

Inženjerstvo promptova 246
Lanac razmišljanja 247
Promena režima 248
Dodela uloge 249
Prompt Object 250
Šablon upita 251
Strukturirani UI/IZ 252
Ulančavanje promptova 253
Prepisivač Promptova 254
Ograđivanje odgovora 255
Analizator upita 256
Prepisivač upita 257
Ventrilokvist 258

Diskretne komponente **259**
Predikat 260
API Fasada 261
Interpreter rezultata 263
Virtuelna mašina 264
Specifikacija i Testiranje 264

Čovek u petlji (HITL) **266**
Obrasci visokog nivoa 266
Eskalacija 267
Povratna sprega 268
Pasivno zračenje informacija 269
Kolaborativno donošenje odluka (CDM) 271
Kontinuirano učenje 272
Etička razmatranja 272
Tehnološki napredak i budući izgledi 272

Inteligentno rukovanje greškama **274**
Tradicionalni pristupi rukovanju greškama 274
Kontekstualna dijagnostika grešaka 275
Inteligentno Izveštavanje o Greškama 276
Prediktivno sprečavanje grešaka 277
Pametni Oporavak od Grešaka 277
Personalizovana komunikacija o greškama 278
Adaptivni tok rada za rukovanje greškama 279

Kontrola kvaliteta **280**
Eval 281
Guardrail 283
Zaštitne mere i Evaluacije: Dve strane istog novčića 283

Rečnik pojmova **285**
Rečnik pojmova 285

Index **290**

Predgovor Gregora Hohpea

Predgovor

O knjizi

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

O primerima koda

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

Šta ne obrađujem

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

Kome je namenjena ova knjiga

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

Izgradnja zajedničkog rečnika

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

Uključivanje

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

Zahvalnice

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

Šta je sa ilustracijama?

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

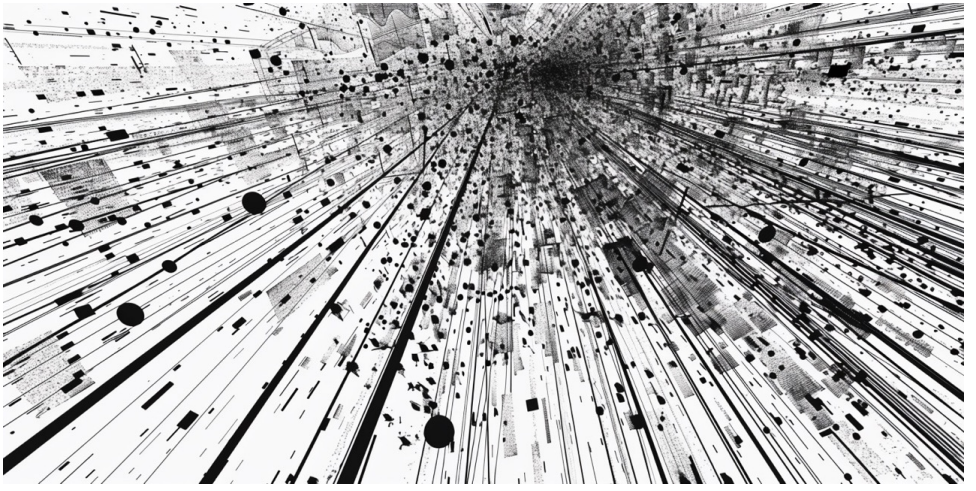
O Lean Publishing-u

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

O Autoru

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

Uvod



Ako jedva čekate da počnete da integrišete AI velike jezičke modele (VJM) u vaše programerske projekte, slobodno pređite odmah na obrasce i primere koda predstavljene u kasnijim poglavljima. Međutim, da biste u potpunosti cenili snagu i potencijal ovih obrazaca, vredi zastati na trenutak da razumete širi kontekst i kohezivni pristup koji oni predstavljaju.

Obrasci nisu samo kolekcija izolovanih tehnika, već jedinstveni okvir za integraciju AI-ja u vaše aplikacije. Ja koristim Ruby on Rails, ali ovi obrasci bi trebalo da funkcionišu u praktično bilo kom drugom programskom okruženju. Oni se bave širokim spektrom pitanja, od upravljanja podacima i optimizacije performansi do korisničkog iskustva i bezbednosti, pružajući sveobuhvatni skup alata za unapređenje tradicionalnih programerskih praksi mogućnostima AI-ja.

Svaka kategorija obrazaca se bavi specifičnim izazovom ili prilikom koja se javlja kada se AI komponente ugrađuju u vašu aplikaciju. Razumevanjem odnosa i sinergije između

ovih obrazaca, možete donositi informisane odluke o tome gde i kako najefikasnije primeniti AI.

Obrasci nikada nisu preskriptivna rešenja i ne treba ih tako tretirati. Oni su zamišljeni kao prilagodljivi gradivni blokovi koje treba prilagoditi jedinstvenim zahtevima i ograničenjima vaše sopstvene jedinstvene aplikacije. Uspešna primena ovih obrazaca (kao i bilo kojih drugih u oblasti softvera) oslanja se na duboko razumevanje problemskog domena, potreba korisnika i celokupne tehničke arhitekture vašeg projekta.

Razmišljanja o softverskoj arhitekturi

Počeo sam da programiram 1980-ih i bio sam uključen u hakersku scenu, i nikada nisam izgubio svoj hakerski mentalitet, čak i nakon što sam postao profesionalni programer. Od početka sam uvek imao zdravu dozu skepticizma prema tome kakvu vrednost softverski arhitekti u svojim kućama od slonovače zapravo donose.

Jedan od razloga zbog kojih sam lično toliko uzbuđen zbog promena koje donosi ovaj moćni novi talas AI tehnologije je njegov uticaj na ono što smatramo odlukama *softverske arhitekture*. On dovodi u pitanje tradicionalne predstave o tome šta čini "ispravan" način dizajniranja i implementacije naših softverskih projekata. Takođe dovodi u pitanje da li se arhitektura i dalje može posmatrati prvenstveno kao *delovi sistema koje je teško promeniti*, s obzirom na to da AI unapređenje čini lakšim nego ikad da se promeni bilo koji deo vašeg projekta, u bilo kom trenutku.

Možda ulazimo u vrhunske godine "postmodernog" pristupa softverskom inženjerstvu. U ovom kontekstu, postmoderno se odnosi na fundamentalni pomak od tradicionalnih paradigmi, gde su programeri bili odgovorni za pisanje i održavanje svake linije koda. Umesto toga, prihvata se ideja delegiranja zadataka, kao što su manipulacija podacima, složeni algoritmi, pa čak i čitavi delovi aplikacione logike, bibliotekama trećih strana i eksternim API-jima. Ovaj postmoderni pomak predstavlja značajno odstupanje

od konvencionalne mudrosti izgradnje aplikacija od nule i izaziva programere da preispitaju svoju ulogu u procesu razvoja.

Uvek sam verovao da dobri programeri pišu samo kod koji je apsolutno neophodno napisati, na osnovu učenja Larry Wall-a i drugih hakerskih luminara poput njega. Minimiziranjem količine napisanog koda, možemo se kretati brže, smanjiti površinu za bagove, pojednostaviti održavanje i poboljšati ukupnu pouzdanost naših aplikacija. Manje koda nam omogućava da se fokusiramo na osnovnu poslovnu logiku i korisničko iskustvo, dok delegiramo ostali posao drugim servisima.

Sada kada AI sistemi mogu da se nose sa zadacima koji su ranije bili ekskluzivni domen koda koji su pisali ljudi, trebalo bi da možemo biti još produktivniji i agilniji, sa većim fokusom nego ikad na stvaranje poslovne vrednosti i korisničkog iskustva.

Naravno, postoje kompromisi u delegiranju ogromnih delova vašeg projekta AI sistemima, kao što su potencijalni gubitak kontrole i potreba za robusnim mehanizmima praćenja i povratnih informacija. Zato je potreban novi set veština i znanja, uključujući bar neko osnovno razumevanje načina na koji AI funkcioniše.

Šta je Veliki jezički model?

Veliki jezički modeli (VJM) su vrsta modela veštačke inteligencije koji su privukli značajnu pažnju poslednjih godina, još od lansiranja GPT-3 od strane OpenAI 2020. godine. VJM-ovi su dizajnirani da obrađuju, razumeju i generišu ljudski jezik sa izuzetnom preciznošću i tečnošću. U ovom odeljku, ukratko ćemo pogledati kako VJM--ovi funkcionišu i zašto su pogodni za izgradnju inteligentnih sistemskih komponenti.

U svojoj suštini, VJM-ovi su zasnovani na algoritmima dubokog učenja, konkretno neuralnim mrežama. Ove mreže se sastoje od međusobno povezanih čvorova, ili neurona, koji obrađuju i prenose informacije. Arhitektura izbora za VJM-ove je često Transformerski model, koji se pokazao kao veoma efikasan u rukovanju sekvencijalnim podacima poput teksta.

Transformer modeli se zasnivaju na mehanizmu pažnje i prvenstveno se koriste za zadatke koji uključuju sekvencijalne podatke, poput obrade prirodnog jezika. Transformeri obrađuju ulazne podatke odjednom umesto sekvencijalno, što im omogućava da efikasnije uhvate zavisnosti dugog dometa. Oni imaju slojeve mehanizama pažnje koji pomažu modelu da se fokusira na različite delove ulaznih podataka kako bi razumeo kontekst i odnose.

Proces obuke velikih jezičkih modela uključuje izlaganje modela ogromnim količinama tekstualnih podataka, kao što su knjige, članci, veb stranice i repozitorijumi koda. Tokom obuke, model uči da prepoznaje obrasce, odnose i strukture unutar teksta. On hvata statističke osobine jezika, kao što su gramatička pravila, asocijacije reči i kontekstualna značenja.

Jedna od ključnih tehnika koja se koristi u obuci velikih jezičkih modela je nenadgledano učenje. To znači da model uči iz podataka bez eksplicitnog označavanja ili vođenja. Sam otkriva obrasce i reprezentacije analizirajući zajedničko pojavljivanje reči i fraza u podacima za obuku. Ovo omogućava velikim jezičkim modelima da razviju duboko razumevanje jezika i njegovih složenosti.

Još jedan važan aspekt velikih jezičkih modela je njihova sposobnost da upravljaju *kontekstom*. Prilikom obrade teksta, veliki jezički modeli uzimaju u obzir ne samo pojedinačne reči već i okolni kontekst. Oni uzimaju u obzir prethodne reči, rečenice, pa čak i pasuse kako bi razumeli značenje i nameru teksta. Ovo kontekstualno razumevanje omogućava velikim jezičkim modelima da generišu koherentne i relevantne odgovore. Jedan od glavnih načina na koji procenjujemo sposobnosti određenog modela velikog jezičkog modela je razmatranje veličine konteksta koji mogu uzeti u obzir pri generisanju odgovora.

Nakon obuke, veliki jezički modeli se mogu koristiti za širok spektar jezičkih zadataka. Mogu generisati tekst nalik ljudskom, odgovarati na pitanja, sumirati dokumente, prevoditi jezike, pa čak i pisati kod. Svestranost velikih jezičkih modela čini ih vrednim za izgradnju inteligentnih sistemskih komponenti koje mogu komunicirati sa

korisnicima, obrađivati i analizirati tekstualne podatke i generisati smislene izlaze.

Uključivanjem velikih jezičkih modela u arhitekturu aplikacije, možete kreirati AI komponente koje razumeju i obrađuju korisnički unos, generišu dinamički sadržaj i pružaju inteligentne preporuke ili akcije. Ali rad sa velikim jezičkim modelima zahteva pažljivo razmatranje potrebnih resursa i kompromisa performansi. Veliki jezički modeli su računski intenzivni i mogu zahtevati značajnu procesorsku snagu i memoriju (drugim rečima, novac) za rad. Većina nas će morati proceniti troškovne implikacije integracije velikih jezičkih modela u naše aplikacije i postupiti u skladu s tim.

Razumevanje zaključivanja

Zaključivanje se odnosi na proces kojim model generiše predviđanja ili izlaze na osnovu novih, neviđenih podataka. To je faza u kojoj se obučeni model koristi za donošenje odluka ili generisanje teksta, slika ili drugog sadržaja kao odgovor na korisničke unose.

Tokom faze obuke, AI model uči iz velikog skupa podataka prilagođavanjem svojih parametara kako bi minimizirao grešku u svojim predviđanjima. Nakon obuke, model može primeniti ono što je naučio na nove podatke. Zaključivanje je način na koji model koristi svoje naučene obrasce i znanje za generisanje izlaza.

Za velike jezičke modele, zaključivanje uključuje uzimanje upita ili ulaznog teksta i proizvođenje koherentnog i kontekstualno relevantnog odgovora, kao tok *tokena* (o kojima ćemo uskoro govoriti). To može biti odgovaranje na pitanje, dovršavanje rečenice, generisanje priče ili prevođenje teksta, među mnogim drugim zadacima.

Za razliku od načina na koji vi i ja razmišljamo, "razmišljanje" AI modela putem zaključivanja se dešava u jednoj operaciji bez stanja. To jest, njegovo razmišljanje je ograničeno na proces generisanja. Bukvalno mora razmišljati naglas, kao da sam vas pitao pitanje i prihvatio odgovor samo u stilu "toka svesti".

Veliki jezički modeli dolaze u mnogim veličinama i ukusima

Iako su praktično svi popularni veliki jezički modeli zasnovani na istoj osnovnoj transformer arhitekturi i obučeni na ogromnim tekstualnim skupovima podataka, dolaze u različitim veličinama i fino su podešeni za različite svrhe. Veličina velikog jezičkog modela, merena brojem parametara u njegovoj neuralnoj mreži, ima veliki uticaj na njegove sposobnosti. Veći modeli sa više parametara, poput GPT-4, za koji se govori da ima 1 do 2 triliona parametara, generalno su informisaniji i sposobniji od manjih modela. Međutim, veći modeli takođe zahtevaju mnogo više računarske snage za rad, što se prevodi u veći trošak kada ih koristite putem API poziva.

Da bi veliki jezički modeli bili praktičniji i prilagođeni specifičnim slučajevima upotrebe, osnovni modeli se često fino podešavaju na ciljanijim skupovima podataka. Na primer, veliki jezički model može biti obučen na velikom korpusu dijaloga da bi se specijalizovao za konverzacionu veštačku inteligenciju. Drugi su obučeni na kodu da bi im se usadilo programersko znanje. Postoje čak i modeli koji su posebno obučeni za interakcije sa korisnicima u stilu igranja uloga!

Modeli zasnovani na pretraživanju naspram generativnih modela

U svetu velikih jezičkih modela (LLM), postoje dva glavna pristupa generisanju odgovora: modeli zasnovani na pretraživanju i generativni modeli. Svaki pristup ima svoje prednosti i mane, a razumevanje razlika između njih može vam pomoći da odaberete pravi model za vaš specifični slučaj upotrebe.

Modeli zasnovani na pretraživanju

Modeli zasnovani na pretraživanju, poznati i kao modeli za pronalaženje informacija, generišu odgovore pretragom velike baze postojećeg teksta i odabirom najrelevantnijih pasusa na osnovu ulaznog upita. Ovi modeli ne generišu novi tekst iz početka, već spajaju delove iz baze podataka kako bi formirali koherentan odgovor.

Jedna od glavnih prednosti modela zasnovanih na pretraživanju je njihova sposobnost da pruže činjenično tačne i ažurne informacije. Pošto se oslanjaju na bazu kuriranog teksta, mogu da izvuku relevantne informacije iz pouzdanih izvora i predstave ih korisniku. Ovo ih čini pogodnim za aplikacije koje zahtevaju precizne, činjenične odgovore, kao što su sistemi za odgovaranje na pitanja ili baze znanja.

Međutim, modeli zasnovani na pretraživanju imaju određena ograničenja. Oni su dobri onoliko koliko je dobra baza podataka koju pretražuju, tako da kvalitet i pokrivenost baze podataka direktno utiču na performanse modela. Dodatno, ovi modeli mogu imati poteškoća sa generisanjem koherentnih i prirodno zvučećih odgovora, jer su ograničeni na tekst dostupan u bazi podataka.

U ovoj knjizi ne obrađujemo upotrebu čistih modela za pretraživanje.

Generativni modeli

Generativni modeli, s druge strane, kreiraju novi tekst iz početka na osnovu obrazaca i veza koje su naučili tokom treninga. Ovi modeli koriste svoje razumevanje jezika da generišu nove odgovore koji su prilagođeni ulaznom upitu.

Glavna prednost generativnih modela je njihova sposobnost da proizvode kreativan, koherentan i kontekstualno relevantan tekst. Mogu da vode otvorene razgovore, generišu priče, pa čak i pišu kod. Ovo ih čini idealnim za aplikacije koje zahtevaju otvorenije i dinamičnije interakcije, kao što su četbotovi, kreiranje sadržaja i asistenti za kreativno pisanje.

Međutim, generativni modeli ponekad mogu proizvesti nedosledne ili činjenično netačne informacije, jer se oslanjaju na obrasce naučene tokom treninga umesto na kuriranu bazu činjenica. Takođe mogu biti skloniji pristrasnostima i halucinacijama, generišući tekst koji zvuči uverljivo ali nije nužno tačan.

Primeri generativnih LLM-ova uključuju OpenAI-jev GPT niz (GPT-3, GPT-4) i Anthropic-ov Claude.

Hibridni modeli

Nekoliko komercijalno dostupnih LLM-ova kombinuje i pristup pretraživanja i generativni pristup u hibridnom modelu. Ovi modeli koriste tehnike pretraživanja da pronađu relevantne informacije iz baze podataka, a zatim koriste generativne tehnike da sintetizuju te informacije u koherentan odgovor.

Hibridni modeli teže da kombinuju činjeničnu tačnost modela zasnovanih na pretraživanju sa sposobnostima generisanja prirodnog jezika generativnih modela. Oni mogu pružiti pouzdanije i ažurnije informacije, zadržavajući pri tome sposobnost vođenja otvorenih razgovora.

Pri izboru između modela zasnovanih na pretraživanju i generativnih modela, trebalo bi da razmotrite specifične zahteve vaše aplikacije. Ako je primarni cilj pružanje tačnih, činjeničnih informacija, model zasnovan na pretraživanju može biti najbolji izbor. Ako aplikacija zahteva otvorenije i kreativnije interakcije, generativni model može biti pogodniji. Hibridni modeli nude ravnotežu između dva pristupa i mogu biti dobar izbor za aplikacije koje zahtevaju i činjeničnu tačnost i generisanje prirodnog jezika.

Na kraju, izbor između modela zasnovanih na pretraživanju i generativnih modela zavisi od specifičnog slučaja upotrebe i kompromisa između tačnosti, kreativnosti i fleksibilnosti. Razumevanjem prednosti i ograničenja svakog pristupa možete donositi informisane odluke.

Podešavanje putem instrukcija

Podešavanje putem instrukcija je podskup konverzacijskog treninga koji koristi upite i odgovore koje su napisali ljudi kako bi se LLM-ovi bolje osposobili za razumevanje i praćenje instrukcija. Evo nekoliko jednostavnih primera koji ilustruju kako podešavanje putem instrukcija na upitima i odgovorima koje su napisali ljudi čini LLM-ove boljim u razumevanju i praćenju instrukcija:

Primer 1: Korisnička podrška

Instrukcija: "Dajte odgovor kupcu koji traži povraćaj novca."

Upit koji je napisao čovek: "Želeo/la bih da zatražim povraćaj novca za svoju nedavnu kupovinu. Proizvod nije ispunio moja očekivanja."

Odgovor koji je napisao čovek: "Žao nam je što proizvod nije ispunio vaša očekivanja. Da bismo obradili vaš zahtev za povraćaj novca, molimo vas da dostavite broj porudžbine i razlog za povraćaj. Pregledaćemo vaš zahtev i javiti vam se u roku od 2-3 radna dana."

Primer 2: Obrazovna pomoć

Instrukcija: "Objasnite koncept fotosinteze desetogodišnjaku."

Upit koji je napisao čovek: "Možeš li da objasniš kako biljke proizvode svoju hranu?"

Odgovor koji je napisao čovek: "Naravno! Biljke proizvode svoju hranu kroz proces koji se zove fotosinteza. Koriste sunčevu svetlost, vodu i ugljen-dioksid da stvaraju hranu i kiseonik. Lišće biljke ima poseban deo koji se zove hlorofil koji im pomaže da zarobe sunčevu svetlost."

Primer 3: Lični Asistent

> Instrukcija: “Zakaži sastanak za sledeći utorak u 15h.”
>
> Upit koji je napisao čovek: “Možeš li da zakažeš sastanak za utorak u 15h?”
>
> Odgovor koji je napisao čovek: “Naravno, zakazao sam vam sastanak za sledeći utorak u 15h. Da li vam je potrebno još nešto?”

Rezultat je raznovrstan ekosistem LLM-ova različitih veličina i sa različitim specijalizacijama. Manji modeli u opsegu od 1-7 milijardi parametara pružaju dobre opšte jezičke sposobnosti dok su efikasniji za pokretanje.

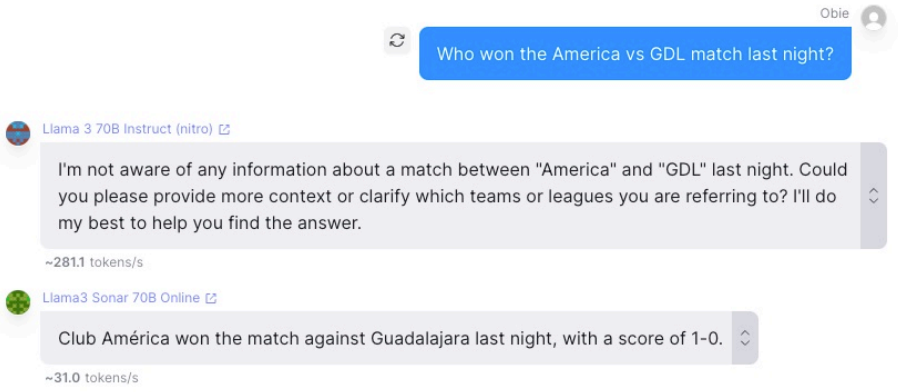
- Mistral 7B
- Llama 3 8B
- Gemma 7B

Modeli srednje veličine od oko 30-70 milijardi parametara nude jače sposobnosti rezonovanja i praćenja instrukcija.

- Llama 3 70B
- Qwen2 70B
- Mixtral 8x22B

Kada birate LLM za integraciju u aplikaciju, morate uravnotežiti mogućnosti modela sa praktičnim faktorima kao što su cena, kašnjenje, dužina konteksta i filtriranje sadržaja. Manji modeli, obučeni na instrukcijama, često su najbolji izbor za jednostavnije jezičke zadatke, dok najveći modeli mogu biti potrebni za složeno rezonovanje ili analizu. Podaci za obuku modela su takođe važna stavka, jer određuju datum preseka znanja modela.

Određeni modeli, poput nekih od Perplexity-ja, povezani su sa izvorima informacija u realnom vremenu, tako da efektivno nemaju datum preseka. Kada im postavite pitanja, mogu samostalno odlučiti da pretražuju internet i preuzimaju proizvoljne veb stranice kako bi generisali odgovor.

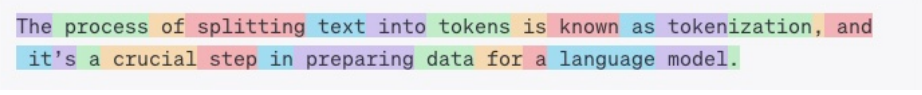


Slika 1. Llama3 sa i bez pristupa internetu

Na kraju, ne postoji univerzalni LLM. Razumevanje varijacija u veličini modela, arhitekturi i obuci je ključno za odabir pravog modela za dati slučaj upotrebe. Eksperimentisanje sa različitim modelima je jedini praktičan način da se otkrije koji pružaju najbolje performanse za zadati zadatak.

Tokenizacija: Razbijanje teksta na delove

Pre nego što veliki jezički model može da obradi tekst, taj tekst treba da bude razložen na manje jedinice koje se zovu *tokeni*. Tokeni mogu biti pojedinačne reči, delovi reči, pa čak i pojedinačni karakteri. Proces deljenja teksta na tokene poznat je kao tokenizacija, i to je ključni korak u pripremi podataka za jezički model.



Slika 2. Ova rečenica sadrži 27 tokena

Različiti LLM-ovi koriste različite strategije tokenizacije, što može značajno uticati na performanse i mogućnosti modela. Neki uobičajeni tokenizatori koje koriste LLM-ovi

uključuju:

- **GPT (Kodiranje parova bajtova)**: GPT tokenizatori koriste tehniku koja se zove kodiranje parova bajtova (BPE) za razbijanje teksta na jedinice podreči. BPE iterativno spaja najčešće parove bajtova u tekstualnom korpusu, formirajući rečnik tokena podreči. Ovo omogućava tokenizatoru da obrađuje retke i nove reči tako što ih razlaže na češće delove podreči. GPT tokenizatore koriste modeli poput GPT-3 i GPT-4.

- **Llama (SentencePiece)**: Llama tokenizatori koriste SentencePiece biblioteku, koja je nenadgledani tokenizator i detokenizator teksta. SentencePiece tretira ulazni tekst kao niz Unicode karaktera i uči vokabular podreči na osnovu korpusa za obuku. Može da obrađuje bilo koji jezik koji se može kodirati u Unicode-u, što ga čini pogodnim za višejezične modele. Llama tokenizatore koriste modeli poput Meta-inog Llama i Alpaca.

- **SentencePiece (Unigram)**: SentencePiece tokenizatori takođe mogu koristiti drugačiji algoritam pod nazivom Unigram, koji se zasniva na tehnici regularizacije podreči. Unigram tokenizacija određuje optimalni vokabular podreči na osnovu unigram jezičkog modela, koji dodeljuje verovatnoće pojedinačnim jedinicama podreči. Ovaj pristup može proizvesti semantički smislenije podreči u poređenju sa BPE. SentencePiece sa Unigram-om koriste modeli poput Google-ovog T5 i BERT.

- **Google Gemini (Multimodalna Tokenizacija)**: Google Gemini koristi šemu tokenizacije dizajniranu za rukovanje različitim tipovima podataka, uključujući tekst, slike, audio, video zapise i kod. Ova multimodalna sposobnost omogućava Gemini-ju da obrađuje i integriše različite oblike informacija. Posebno je značajno da Google Gemini 1.5 Pro ima kontekstni prozor koji može da obradi milione tokena, što je mnogo više od prethodnih modela. Ovaj obimni kontekstni prozor

omogućava modelu da obradi veći kontekst, potencijalno vodeći do preciznijih odgovora. Međutim, važno je napomenuti da je Gemini-jeva šema tokenizacije mnogo bliža jednom tokenu po karakteru nego kod drugih modela. To znači da stvarni trošak korišćenja Gemini modela može biti značajno veći nego što se očekuje ako ste navikli na korišćenje modela poput GPT-a, jer se Google-ovo određivanje cena zasniva na karakterima umesto na tokenima.

Izbor tokenizatora utiče na nekoliko aspekata LLM-a, uključujući:

- **Veličina vokabulara**: Tokenizator određuje veličinu vokabulara modela, što je skup jedinstvenih tokena koje prepoznaje. Veći, detaljniji vokabular može pomoći modelu da obradi širi spektar reči i fraza i čak postane multimodalan (sposoban za razumevanje i generisanje više od samog teksta), ali takođe povećava memorijske zahteve modela i računarsku složenost.
- **Rukovanje retkim i nepoznatim rečima**: Tokenizatori koji koriste jedinice podreči, poput BPE i SentencePiece, mogu rastaviti retke i nepoznate reči na češće delove podreči. Ovo omogućava modelu da napravi obrazovane pretpostavke o značenju reči koje ranije nije video, na osnovu podreči koje sadrže.
- **Višejezična podrška**: Tokenizatori poput SentencePiece-a, koji mogu da rukuju bilo kojim jezikom koji se može kodirati u Unicode-u, pogodni su za višejezične modele koji treba da obrađuju tekst na više jezika.

Prilikom odabira LLM-a za određenu primenu, važno je razmotriti tokenizator koji koristi i koliko se dobro uklapa sa specifičnim potrebama obrade jezika za dati zadatak. Tokenizator može imati značajan uticaj na sposobnost modela da rukuje terminologijom specifičnom za domen, retkim rečima i višejezičnim tekstom.

Veličina Konteksta: Koliko Informacija Jezički Model Može Koristiti Tokom Zaključivanja?

Kada se govori o jezičkim modelima, veličina konteksta se odnosi na količinu teksta koju model može da razmotri prilikom obrade ili generisanja svojih odgovora. To je u suštini mera koliko informacija model može da "zapamti" i koristi za informisanje svojih izlaza (izraženo u tokenima). Veličina konteksta jezičkog modela može imati značajan uticaj na njegove mogućnosti i vrste zadataka koje može efikasno da obavlja.

Šta je Veličina Konteksta?

U tehničkom smislu, veličina konteksta je određena brojem tokena (reči ili delova reči) koje jezički model može da obradi u jednom ulaznom nizu. Ovo se često naziva "opsegom pažnje" ili "kontekstnim prozorom" modela. Što je veća veličina konteksta, više teksta model može odjednom da razmotri prilikom generisanja odgovora ili izvršavanja zadatka.

Različiti jezički modeli imaju različite veličine konteksta, u rasponu od nekoliko stotina tokena do miliona tokena. Za referencu, tipičan pasus teksta može sadržati oko 100-150 tokena, dok cela knjiga može sadržati desetine ili stotine hiljada tokena.

> Postoje čak i radovi o efikasnim metodama za skaliranje Transformer-zasnovanih Velikih Jezičkih Modela (LLM) za beskonačno duge unose sa ograničenom memorijom i računanjem.

Zašto je veličina konteksta važna?

Veličina konteksta jezičkog modela ima značajan uticaj na njegovu sposobnost da razume i generiše koherentan, kontekstualno relevantan tekst. Evo nekoliko ključnih razloga zašto je veličina konteksta bitna:

1. **Razumevanje dugačkih sadržaja**: Modeli sa većim kontekstom mogu bolje da razumeju i analiziraju duže tekstove, kao što su članci, izveštaji, ili čak cele knjige. Ovo je ključno za zadatke poput sumiranja dokumenata, odgovaranja na pitanja i analize sadržaja.

2. **Održavanje koherentnosti**: Veći kontekstni prozor omogućava modelu da održi koherentnost i doslednost kroz duže delove izlaznog teksta. Ovo je važno za zadatke poput generisanja priča, sistema za dijalog i kreiranje sadržaja, gde je održavanje doslednog narativa ili teme od suštinskog značaja. Takođe je apsolutno ključno kada se VJM koriste za generisanje ili transformaciju strukturiranih podataka.
3. **Hvatanje zavisnosti dugog dometa**: Neki jezički zadaci zahtevaju razumevanje odnosa između reči ili fraza koje su međusobno udaljene u tekstu. Modeli sa većim kontekstom su bolje opremljeni za hvatanje ovih zavisnosti dugog dometa, što može biti važno za zadatke poput analize sentimenta, prevođenja i razumevanja jezika.
4. **Rukovanje složenim instrukcijama**: U aplikacijama gde se jezički modeli koriste za praćenje složenih instrukcija u više koraka, veći kontekst omogućava modelu da razmotri celokupan set instrukcija pri generisanju odgovora, umesto samo nekoliko najskorijih reči.

Primeri jezičkih modela sa različitim veličinama konteksta

Evo nekoliko primera jezičkih modela sa različitim veličinama konteksta:

- OpenAI GPT-3.5 Turbo: 4.095 tokena
- Mistral 7B Instruct: 32.768 tokena
- Anthropic Claude v1: 100.000 tokena
- OpenAI GPT-4 Turbo: 128.000 tokena
- Anthropic Claude v2: 200.000 tokena

- Google Gemini Pro 1.5: 2,8M tokena

Kao što možete videti, postoji širok raspon veličina konteksta među ovim modelima, od oko 4.000 tokena za OpenAI GPT-3.5 Turbo model do 200.000 tokena za Anthropic Claude v2 model. Neki modeli, poput Google-ovog PaLM 2 i OpenAI-evog GPT-4, nude različite varijante sa većim kontekstom (npr. "32k" verzije), koje mogu da rukuju još dužim ulaznim sekvencama. A trenutno (april 2024.) Google Gemini Pro se hvali sa skoro 3 miliona tokena!

Vredno je napomenuti da veličina konteksta može varirati u zavisnosti od specifične implementacije i verzije određenog modela. Na primer, originalni OpenAI GPT-4 model ima veličinu konteksta od 8.191 tokena, dok kasnije GPT-4 varijante kao što su Turbo i 4o imaju mnogo veću veličinu konteksta od 128.000 tokena.

> Sam Altman je uporedio trenutna kontekstna ograničenja sa kilobajtima radne memorije sa kojima su programeri personalnih računara morali da se bore 80-ih, i rekao da ćemo u bliskoj budućnosti moći da smestimo "sve vaše lične podatke" u kontekst velikog jezičkog modela.

Izbor odgovarajuće veličine konteksta

Prilikom odabira jezičkog modela za određenu aplikaciju, važno je razmotriti zahteve za veličinom konteksta za dati zadatak. Za zadatke koji uključuju kratke, izolovane delove teksta, poput analize sentimenta ili jednostavnog odgovaranja na pitanja, manja veličina konteksta može biti dovoljna. Međutim, za zadatke koji zahtevaju razumevanje i generisanje dužih, složenijih tekstova, veća veličina konteksta će verovatno biti neophodna.

Vredno je napomenuti da veće veličine konteksta često dolaze sa povećanim računarskim troškovima i sporijim vremenom obrade, jer model mora da razmotri više informacija

pri generisanju odgovora. Kao takvi, morate postići ravnotežu između veličine konteksta i performansi pri izboru jezičkog modela za vašu aplikaciju.

> Zašto ne bismo jednostavno izabrali model sa najvećim kontekstom i napunili ga sa što više informacija? Pa, pored faktora performansi, druga glavna stavka je cena. U martu 2024. jedan *jedini* ciklus upita i odgovora koristeći Google Gemini Pro 1.5 sa punim kontekstom će vas koštati skoro 8 dolara (USD). Ako imate slučaj upotrebe koji opravdava taj trošak, svaka čast! Ali za većinu aplikacija, to je jednostavno preskupo za nekoliko redova veličine.

Pronalaženje Igala u Plastu Sena

Koncept pronalaženja igle u plastu sena dugo je bio metafora za izazove pretraživanja u velikim skupovima podataka. U domenu velikih jezičkih modela, malo modifikujemo ovu analogiju. Zamislite da ne tražimo samo jednu činjenicu zakopanu unutar ogromnog teksta (poput cele antologije eseja Paula Grahama), već više činjenica rasutih svuda. Ovaj scenario više podseća na pronalaženje nekoliko igala u prostranom polju, ne samo u jednom plastu sena. Evo štos: ne samo da moramo locirati te igle, već moramo i da ih upletemo u koherentnu nit.

Kada se suoče sa zadatkom pronalaženja i rezonovanja o višestrukim činjenicama ugrađenim u dugačke kontekste, veliki jezički modeli se suočavaju sa dvostrukim izazovom. Prvo, postoji jednostavan problem tačnosti pronalaženja--ona prirodno opada kako se broj činjenica povećava. Ovo je očekivano; na kraju krajeva, praćenje više detalja kroz obiman tekst opterećuje čak i najsofisticiranije modele.

Drugo, i možda kritičnije, je izazov rezonovanja sa ovim činjenicama. Jedno je izdvojiti činjenice; sasvim drugo je sintetizovati ih u koherentnu priču ili odgovor. Tu dolazi pravi test. Performanse velikih jezičkih modela u zadacima rezonovanja imaju tendenciju da

se pogoršavaju više nego u jednostavnim zadacima pronalaženja. Ova degradacija nije samo pitanje obima; radi se o složenom plesu konteksta, relevantnosti i zaključivanja.

Zašto se ovo dešava? Pa, razmotrite dinamiku pamćenja i pažnje u ljudskoj kogniciji, koja se u određenoj meri ogleda i u velikim jezičkim modelima. Prilikom obrade velike količine informacija, veliki jezički modeli, poput ljudi, mogu izgubiti trag ranijih detalja dok apsorbuju nove. Ovo je posebno tačno kod modela koji nisu eksplicitno dizajnirani da automatski prioritizuju ili se vraćaju na ranije segmente teksta.

Štaviše, sposobnost velikog jezičkog modela da uplete ove pronađene činjenice u koherentan odgovor slična je izgradnji narativa. Ovo zahteva ne samo pronalaženje informacija već i duboko razumevanje i kontekstualno pozicioniranje, što ostaje ozbiljan izazov za trenutnu veštačku inteligenciju.

Dakle, šta ovo znači za nas kao programere i integratore ovih tehnologija? Moramo biti izuzetno svesni ovih ograničenja kada dizajniramo sisteme koji se oslanjaju na velike jezičke modele za rukovanje složenim zadacima sa dugačkim tekstovima. Razumevanje da se performanse mogu pogoršati pod određenim uslovima pomaže nam da postavimo realna očekivanja i razvijemo bolje rezervne mehanizme ili dopunske strategije.

Modaliteti: Izvan Teksta

Dok je većina jezičkih modela danas fokusirana na obradu i generisanje teksta, postoji rastući trend prema multimodalnim modelima koji mogu prirodno primati i proizvoditi više vrsta podataka, kao što su slike, audio i video. Ovi multimodalni modeli otvaraju nove mogućnosti za aplikacije pokretane veštačkom inteligencijom koje mogu razumeti i generisati sadržaj kroz različite modalitete.

Šta su Modaliteti?

U kontekstu jezičkih modela, modaliteti se odnose na različite vrste podataka koje model može da obrađuje i generiše. Najčešći modalitet je tekst, koji uključuje pisani jezik u

različitim oblicima poput knjiga, članaka, veb-sajtova i objava na društvenim mrežama. Međutim, postoji nekoliko drugih modaliteta koji se sve više uključuju u jezičke modele:

- **Slike**: Vizuelni podaci kao što su fotografije, ilustracije i dijagrami.
- **Audio**: Zvučni podaci kao što su govor, muzika i zvukovi iz okruženja.
- **Video**: Pokretni vizuelni podaci, često praćeni zvukom, kao što su video klipovi i filmovi.

Svaki modalitet predstavlja jedinstvene izazove i mogućnosti za jezičke modele. Na primer, slike zahtevaju da model razume vizuelne koncepte i odnose, dok audio zahteva da model obrađuje i generiše govor i druge zvukove.

Multimodalni Jezički Modeli

Multimodalni jezički modeli su dizajnirani da rukuju sa više modaliteta unutar jednog modela. Ovi modeli tipično imaju specijalizovane komponente ili slojeve koji mogu i da razumeju ulazne podatke i da generišu izlazne podatke u različitim modalitetima. Neki značajni primeri multimodalnih jezičkih modela uključuju:

- **OpenAI-jev GPT-4o**: GPT-4o je veliki jezički model koji prirodno razume i obrađuje govorni audio pored teksta. Ova sposobnost omogućava GPT-4o da obavlja zadatke kao što su transkripcija govornog jezika, generisanje teksta iz audio ulaza i pružanje odgovora na osnovu govornih upita.
- **OpenAI-jev GPT-4 sa vizuelnim unosom**: GPT-4 je veliki jezički model koji može da obrađuje i tekst i slike. Kada dobije sliku kao ulaz, GPT-4 može da analizira sadržaj slike i generiše tekst koji opisuje ili odgovara na vizuelne informacije.
- **Google-ov Gemini**: Gemini je multimodalni model koji može da rukuje tekstom, slikama i videom. Koristi jedinstvenu arhitekturu koja omogućava međumodalno razumevanje i generisanje, omogućavajući zadatke poput opisivanja slika, sažimanja videa i odgovaranja na vizuelna pitanja.

- **DALL-E i Stable Diffusion**: Iako nisu jezički modeli u tradicionalnom smislu, ovi modeli demonstriraju moć višemodalne veštačke inteligencije generisanjem slika iz tekstualnih opisa. Oni pokazuju potencijal modela koji mogu da prevode između različitih modaliteta.

Prednosti i primene višemodalnih modela

Višemodalni jezički modeli nude nekoliko prednosti i omogućavaju širok spektar primena, uključujući:

- **Poboljšano razumevanje**: Obradom informacija iz više modaliteta, ovi modeli mogu da steknu sveobuhvatnije razumevanje sveta, slično načinu na koji ljudi uče iz različitih čulnih inputa.
- **Međumodalno generisanje**: Višemodalni modeli mogu da generišu sadržaj u jednom modalitetu na osnovu unosa iz drugog, kao što je kreiranje slike iz tekstualnog opisa ili generisanje video rezimea iz pisanog članka.
- **Pristupačnost**: Višemodalni modeli mogu učiniti informacije pristupačnijim prevođenjem između modaliteta, kao što je generisanje tekstualnih opisa slika za korisnike sa oštećenim vidom ili kreiranje audio verzija pisanog sadržaja.
- **Kreativne primene**: Višemodalni modeli se mogu koristiti za kreativne zadatke poput generisanja umetnosti, muzike ili videa na osnovu tekstualnih promptova, otvarajući nove mogućnosti za umetnike i kreatore sadržaja.

Kako višemodalni jezički modeli nastavljaju da napreduju, verovatno će igrati sve važniju ulogu u razvoju aplikacija zasnovanih na veštačkoj inteligenciji koje mogu da razumeju i generišu sadržaj kroz više modaliteta. Ovo će omogućiti prirodnije i intuitivnije interakcije između ljudi i AI sistema, kao i otključati nove mogućnosti za kreativno izražavanje i širenje znanja.

Ekosistemi provajdera

Kada je reč o uključivanju velikih jezičkih modela (LLM) u aplikacije, imate rastući broj opcija za izbor. Svaki veliki LLM provajder, kao što su OpenAI, Anthropic, Google i Cohere, nudi svoj sopstveni ekosistem modela, API-ja i alata. Izbor pravog provajdera uključuje razmatranje različitih faktora, uključujući cene, performanse, filtriranje sadržaja, privatnost podataka i opcije prilagođavanja.

OpenAI

OpenAI je jedan od najpoznatijih provajdera LLM-ova, sa svojom GPT serijom (GPT-3, GPT-4) koja se široko koristi u različitim aplikacijama. OpenAI nudi API jednostavan za korišćenje koji vam omogućava da lako integrišete njihove modele u aplikacije. Oni pružaju niz modela sa različitim mogućnostima i cenovnim rangovima, od početnog Ada modela do moćnog Davinci modela.

OpenAI-jev ekosistem takođe uključuje alate poput OpenAI Playground-a, koji vam omogućava da eksperimentišete sa promptovima i fino podešavate modele za specifične slučajeve upotrebe. Oni nude opcije za filtriranje sadržaja kako bi pomogli u sprečavanju generisanja neprikladnog ili štetnog sadržaja.

Kada koristim OpenAI modele direktno, oslanjam se na Alex Rudall-ovu ruby-openai biblioteku.

Anthropic

Anthropic je još jedan veliki igrač u LLM prostoru, čiji Claude modeli stiču popularnost zbog snažnih performansi i etičkih razmatranja. Anthropic se fokusira na razvoj bezbednih i odgovornih AI sistema, sa snažnim naglaskom na filtriranje sadržaja i izbegavanje štetnih izlaza.

Anthropic-ov ekosistem uključuje Claude API, koji vam omogućava da integrišete model u svoje aplikacije, kao i alate za inženjerstvo promptova i fino podešavanje. Oni

takođe nude Claude Instant model, koji uključuje mogućnosti veb pretrage za ažurnije i činjenično tačnije odgovore.

Kada koristim Anthropic-ove modele direktno, oslanjam se na Alex Rudall-ovu anthrophic biblioteku.

Google

Google je razvio nekoliko moćnih LLM-ova, uključujući Gemini, BERT, T5 i PaLM. Ovi modeli su poznati po svojim snažnim performansama u širokom spektru zadataka obrade prirodnog jezika. Google-ov ekosistem uključuje TensorFlow i Keras biblioteke, koje pružaju alate i okvire za izgradnju i treniranje modela mašinskog učenja.

Google takođe nudi Cloud AI Platform, koji vam omogućava da lako implementirate i skalirate njihove modele u oblaku. Oni pružaju niz unapred treniranih modela i API-ja za zadatke poput analize sentimenta, prepoznavanja entiteta i prevođenja.

Meta

Meta, ranije poznat kao Facebook, duboko je uključen u razvoj velikih jezičkih modela, što se ističe objavljivanjem modela poput LLaMA i OPT. Ovi modeli se ističu po svojim snažnim performansama u raznovrsnim jezičkim zadacima i uglavnom su dostupni kroz kanale otvorenog koda, podržavajući Meta-inu posvećenost istraživanju i saradnji sa zajednicom.

Meta-in ekosistem je prvenstveno izgrađen oko PyTorch-a, biblioteke za mašinsko učenje otvorenog koda koja je cenjena zbog svojih dinamičkih računarskih mogućnosti i fleksibilnosti, olakšavajući inovativno AI istraživanje i razvoj.

Pored svojih tehničkih ponuda, Meta stavlja snažan naglasak na etički razvoj veštačke inteligencije. Oni implementiraju robusno filtriranje sadržaja i fokusiraju se na smanjenje pristrasnosti, u skladu sa svojim širim ciljevima sigurnosti i odgovornosti u primeni veštačke inteligencije.

Cohere

Cohere je noviji učesnik u prostoru velikih jezičkih modela, fokusirajući se na to da LLM-ovi budu pristupačniji i lakši za korišćenje od konkurencije. Njihov ekosistem uključuje Cohere API, koji pruža pristup nizu prethodno obučenih modela za zadatke poput generisanja teksta, klasifikacije i sumiranja.

Cohere takođe nudi alate za inženjerstvo promptova, fino podešavanje i filtriranje sadržaja. Oni naglašavaju privatnost i bezbednost podataka, sa funkcijama poput šifrovanog skladištenja podataka i kontrole pristupa.

Ollama

Ollama je samohostovana platforma koja korisnicima omogućava da upravljaju i implementiraju različite velike jezičke modele (LLM) lokalno na svojim mašinama, dajući im potpunu kontrolu nad njihovim AI modelima bez oslanjanja na eksterne cloud servise. Ovo podešavanje je idealno za one koji prioritizuju privatnost podataka i žele da svoje AI operacije vode interno.

Platforma podržava niz modela, uključujući verzije Llama, Phi, Gemma i Mistral, koji se razlikuju po veličini i računarskim zahtevima. Ollama olakšava preuzimanje i pokretanje ovih modela direktno iz komandne linije koristeći jednostavne komande poput ollama run <model_name>, i dizajnirana je da radi na različitim operativnim sistemima uključujući macOS, Linux i Windows.

Za programere koji žele da integrišu modele otvorenog koda u svoje aplikacije bez korišćenja udaljenog API-ja, Ollama nudi CLI za upravljanje životnim ciklusom modela slično alatima za upravljanje kontejnerima. Takođe podržava prilagođene konfiguracije i promptove, omogućavajući visok stepen prilagođavanja za specifične potrebe ili slučajeve upotrebe.

Ollama je posebno pogodna za tehnički obrazovane korisnike i programere zbog svog interfejsa komandne linije i fleksibilnosti koju nudi u upravljanju i implementaciji AI

modela. Ovo je čini moćnim alatom za preduzeća i pojedince kojima su potrebne robusne AI mogućnosti bez kompromitovanja sigurnosti i kontrole.

Multi-Model Platforme

Dodatno, postoje provajderi koji hostuju širok spektar modela otvorenog koda, kao što su Together.ai i Groq.. Ove platforme nude fleksibilnost i prilagodljivost, omogućavajući vam da pokrenete i, u nekim slučajevima, čak i fino podesite modele otvorenog koda prema vašim specifičnim potrebama. Na primer, Together.ai pruža pristup nizu LLM-ova otvorenog koda, omogućavajući korisnicima da eksperimentišu sa različitim modelima i konfiguracijama. Groq se fokusira na pružanje ultra visoko-performansnih kompletiranja koja u vreme pisanja ove knjige deluju gotovo magično

Izbor LLM Provajdera

Pri izboru LLM provajdera, trebalo bi razmotriti faktore kao što su:

- **Cene**: Različiti provajderi nude različite modele određivanja cena, od plaćanja po korišćenju do pretplatničkih planova. Važno je razmotriti očekivanu upotrebu i budžet pri odabiru provajdera.
- **Performanse**: Performanse LLM-ova mogu značajno varirati između provajdera, tako da je važno testirati modele na specifičnim slučajevima upotrebe pre donošenja odluke.
- **Filtriranje sadržaja**: U zavisnosti od aplikacije, filtriranje sadržaja može biti kritično razmatranje. Neki provajderi nude robusnije opcije filtriranja sadržaja od drugih.
- **Privatnost podataka**: Ako aplikacija rukuje osetljivim korisničkim podacima, važno je izabrati provajdera sa snažnim praksama privatnosti i bezbednosti podataka.
- **Prilagođavanje**: Neki provajderi nude više fleksibilnosti u pogledu finog podešavanja i prilagođavanja modela za specifične slučajeve upotrebe.

Na kraju, izbor LLM provajdera zavisi od specifičnih zahteva i ograničenja aplikacije. Pažljivim procenjivanjem opcija i razmatranjem faktora poput cena, performansi i privatnosti podataka, možete odabrati provajdera koji najbolje odgovara vašim potrebama.

Takođe je vredno napomenuti da se LLM pejzaž konstantno razvija, sa redovnim pojavljivanjem novih provajdera i modela. Trebalo bi da budete u toku sa najnovijim razvojima i otvoreni za istraživanje novih opcija kako postaju dostupne.

OpenRouter

Kroz ovu knjigu ću se oslanjati isključivo na OpenRouter kao mog izabranog API provajdera. Razlog je jednostavan: to je prodavnica sve-na-jednom-mestu za sve najpopularnije komercijalne modele i modele otvorenog koda. Ako jedva čekate da zaprljate ruke sa AI programiranjem, jedno od najboljih mesta za početak je moja OpenRouter Ruby Biblioteka.

Razmišljanje o performansama

Prilikom integracije jezičkih modela u aplikacije, performanse su ključni faktor koji treba uzeti u obzir. Performanse jezičkog modela mogu se meriti kroz njegovu *latenciju* (vreme potrebno za generisanje odgovora) i *propusnost* (broj zahteva koje može obraditi u jedinici vremena).

Vreme do prvog tokena (TTFT) je još jedna važna metrika performansi, posebno relevantna za chatbotove i aplikacije koje zahtevaju interaktivne odgovore u realnom vremenu. TTFT meri latenciju od trenutka kada je primljen zahtev korisnika do trenutka kada je generisana prva reč (ili token) odgovora. Ova metrika je ključna za održavanje neometanog i privlačnog korisničkog iskustva, jer odloženi odgovori mogu dovesti do frustracije korisnika i smanjenja angažovanja.

Ove metrike performansi mogu imati značajan uticaj na korisničko iskustvo i skalabilnost aplikacije.

Nekoliko faktora može uticati na performanse jezičkog modela, uključujući:

Broj parametara: Veći modeli sa više parametara generalno zahtevaju više računarskih resursa i mogu imati veću latenciju i manju propusnost u poređenju sa manjim modelima.

Hardver: Performanse jezičkog modela mogu značajno varirati u zavisnosti od hardvera na kojem se izvršava. Cloud provajderi nude GPU i TPU instance optimizovane za mašinsko učenje, koje mogu značajno ubrzati zaključivanje modela.

Jedna od dobrih stvari kod OpenRouter-a je što za mnoge modele koje nudi dobijate izbor cloud provajdera sa različitim performansnim profilima i troškovima.

Kvantizacija: Tehnike kvantizacije mogu se koristiti za smanjenje memorijskog otiska i računarskih zahteva modela predstavljanjem težina i aktivacija tipovima podataka niže preciznosti. Ovo može poboljšati performanse bez značajnog žrtvovanja kvaliteta. Kao programer aplikacija, verovatno se nećete baviti treniranjem sopstvenih modela na različitim nivoima kvantizacije, ali dobro je bar biti upoznat sa terminologijom.

Grupisanje: Obrada više zahteva istovremeno u grupama može poboljšati propusnost amortizacijom režijskih troškova učitavanja modela i prenosa podataka.

Keširanje: Keširanje rezultata često korišćenih promptova ili ulaznih sekvenci može smanjiti broj zahteva za zaključivanjem i poboljšati ukupne performanse.

Pri odabiru jezičkog modela za produkcijsku aplikaciju, važno je testirati njegove performanse na reprezentativnim radnim opterećenjima i hardverskim konfiguracijama. Ovo može pomoći u identifikaciji potencijalnih uskih grla i osigurati da model može ispuniti zahtevane performansne ciljeve.

Takođe je vredno razmotriti kompromise između performansi modela i drugih faktora kao što su troškovi, fleksibilnost i lakoća integracije. Na primer, korišćenje manjeg, jeftinijeg modela sa nižom latencijom može biti poželjnije za aplikacije koje zahtevaju odgovore u realnom vremenu, dok veći, moćniji model može biti pogodniji za grupnu obradu ili složene zadatke rezonovanja.

Eksperimentisanje sa različitim LLM modelima

Izbor LLM-a retko je trajna odluka. Kako se novi i poboljšani modeli redovno objavljuju, dobro je graditi aplikacije na modularan način koji omogućava zamenu različitih jezičkih modela tokom vremena. Promptovi i skupovi podataka često se mogu ponovo koristiti kroz različite modele uz minimalne izmene. Ovo vam omogućava da iskoristite najnovija dostignuća u modeliranju jezika bez potrebe za potpunim redizajniranjem aplikacija.

Mogućnost lake zamene između širokog spektra izbora modela je još jedan razlog zašto volim OpenRouter.

Prilikom nadogradnje na novi jezički model, važno je temeljno testirati i validirati njegove performanse i kvalitet izlaza kako bi se osiguralo da ispunjava zahteve aplikacije. Ovo može uključivati ponovno treniranje ili fino podešavanje modela na domenski specifičnim podacima, kao i ažuriranje svih downstream komponenti koje zavise od izlaza modela.

Dizajniranjem aplikacija sa fokusom na performanse i modularnost, možete kreirati skalabilne, efikasne i sisteme otporne na budućnost koji se mogu prilagoditi brzo razvijajućem pejzažu tehnologije modeliranja jezika.

Složeni AI sistemi

Pre zatvaranja našeg uvoda, vredi spomenuti da su pre 2023. godine i eksplozije interesovanja za generativnu AI podstaknutu ChatGPT-om, tradicionalni AI pristupi obično zavisili od integracije pojedinačnih, zatvorenih modela. Nasuprot tome, *Složeni AI sistemi* koriste kompleksne cevovode međusobno povezanih komponenti koje rade zajedno kako bi postigle inteligentno ponašanje.

U svojoj srži, složeni AI sistemi se sastoje od više modula, od kojih je svaki dizajniran da obavlja specifične zadatke ili funkcije. Ovi moduli mogu uključivati generatore, pretraživače, rangere, klasifikatore i razne druge specijalizovane komponente. Razbijanjem celokupnog sistema na manje, fokusirane jedinice, programeri mogu kreirati fleksibilnije, skalabilnije i održivije AI arhitekture.

Jedna od ključnih prednosti složenih AI sistema je njihova sposobnost da kombinuju snage različitih AI tehnika i modela. Na primer, sistem može koristiti veliki jezički model (LLM) za razumevanje i generisanje prirodnog jezika, dok istovremeno koristi zaseban model za preuzimanje informacija ili donošenje odluka zasnovano na pravilima. Ovaj modularni pristup vam omogućava da odaberete najbolje alate i tehnike za svaki specifični zadatak, umesto da se oslanjate na jedinstveno rešenje za sve.

Međutim, izgradnja složenih AI sistema takođe predstavlja jedinstvene izazove. Posebno, osiguravanje ukupne koherentnosti i doslednosti ponašanja sistema zahteva robusno testiranje, praćenje i mehanizme upravljanja.

Pojava moćnih LLM-ova poput GPT-4 nam omogućava da eksperimentišemo sa složenim AI sistemima lakše nego ikada pre, jer su ovi napredni modeli sposobni da upravljaju višestrukim ulogama unutar složenog sistema, kao što su klasifikacija, rangiranje i generisanje, pored svojih sposobnosti razumevanja prirodnog jezika. Ova višestranost omogućava programerima da brzo kreiraju prototipove i iteriraju na arhitekturama složenih AI sistema, otvarajući nove mogućnosti za razvoj inteligentnih aplikacija.

Obrasci implementacije za složene AI sisteme

Složeni AI sistemi mogu se implementirati koristeći različite obrasce, od kojih je svaki dizajniran da odgovori na specifične zahteve i slučajeve upotrebe. Istražimo četiri uobičajena obrasca implementacije: Pitanja i odgovori, Višeagentni/Agentni rešavaoci problema, Konverzacijski AI i CoPiloti.

Pitanja i odgovori

Sistemi za pitanja i odgovore (Q&A) fokusiraju se na pružanje preuzimanja informacija koje je unapređeno sposobnostima razumevanja AI modela kako bi funkcionisali kao više od običnog pretraživača. Kombinovanjem moćnih jezičkih modela sa spoljnim izvorima znanja koristeći Generisanje potpomognuto preuzimanjem (RAG), sistemi za pitanja i odgovore izbegavaju halucinacije i pružaju tačne i kontekstualno relevantne odgovore na upite korisnika.

Ključne komponente Q&A sistema zasnovanog na LLM-u uključuju:

- **Razumevanje i preformulacija upita**: Analiza korisničkih upita i njihovo preformulisanje kako bi se bolje poklopili sa osnovnim izvorima znanja.
- **Preuzimanje znanja**: Preuzimanje relevantnih informacija iz strukturiranih ili nestrukturiranih izvora podataka na osnovu preformulisanog upita.

- **Generisanje odgovora**: Generisanje koherentnih i informativnih odgovora integrisanjem preuzetog znanja sa generativnim sposobnostima jezičkog modela.

RAG podsistemi su posebno važni u Q&A domenima gde je pružanje tačnih i ažurnih informacija ključno, kao što su korisnička podrška, upravljanje znanjem ili obrazovne aplikacije

Višeagentni/Agentni rešavaoci problema

Višeagentni, takođe poznati kao *Agentni*, sistemi sastoje se od više autonomnih agenata koji rade zajedno na rešavanju složenih problema. Svaki agent ima specifičnu ulogu, skup veština i pristup relevantnim alatima ili izvorima informacija. Kroz saradnju i razmenu informacija, ovi agenti mogu da se uhvate u koštac sa zadacima koje bi bilo teško ili nemoguće da jedan agent reši sam.

Ključni principi višeagentnih rešavaoca problema uključuju:

- **Specijalizacija**: Svaki agent se fokusira na specifični aspekt problema, koristeći svoje jedinstvene sposobnosti i znanje.
- **Saradnja**: Agenti komuniciraju i koordiniraju svoje akcije kako bi postigli zajednički cilj, često kroz razmenu poruka ili deljenu memoriju.
- **Prilagodljivost**: Sistem se može prilagoditi promenjenim uslovima ili zahtevima prilagođavanjem uloga i ponašanja pojedinačnih agenata.

Višeagentni sistemi su pogodni za aplikacije koje zahtevaju distribuirano rešavanje problema, kao što su optimizacija lanca snabdevanja, upravljanje saobraćajem ili planiranje reagovanja u vanrednim situacijama

Konverzacijski AI

Konverzacijski AI sistemi omogućavaju interakcije prirodnim jezikom između korisnika i inteligentnih agenata. Ovi sistemi kombinuju razumevanje prirodnog jezika,

upravljanje dijalogom i sposobnosti generisanja jezika kako bi pružili angažujuća i personalizovana konverzacijska iskustva.

Glavne komponente konverzacijskog AI sistema uključuju:

- **Prepoznavanje namere**: Identifikovanje namere korisnika na osnovu njihovog unosa, kao što je postavljanje pitanja, upućivanje zahteva ili izražavanje osećanja.
- **Ekstrakcija entiteta**: Izvlačenje relevantnih entiteta ili parametara iz korisničkog unosa, kao što su datumi, lokacije ili nazivi proizvoda.
- **Upravljanje dijalogom**: Održavanje stanja razgovora, određivanje odgovarajućeg odgovora na osnovu namere korisnika i konteksta, i upravljanje interakcijama koje se odvijaju u više koraka.
- **Generisanje odgovora**: Generisanje odgovora nalik ljudskim koristeći jezičke modele, šablone ili metode zasnovane na preuzimanju.

Konverzacijski AI sistemi se obično koriste u četbotovima za korisničku podršku, virtuelnim asistentima i interfejsima kojima se upravlja glasom. Kao što je ranije pomenuto, većina pristupa, obrazaca i primera koda u ovoj knjizi direktno je izvučena iz mog rada na velikom konverzacijskom AI sistemu pod nazivom Olympia

Kopiloti

Kopiloti su AI asistenti koji rade zajedno sa ljudskim korisnicima kako bi poboljšali njihovu produktivnost i sposobnost donošenja odluka. Ovi sistemi koriste kombinaciju obrade prirodnog jezika, mašinskog učenja i domenskog znanja kako bi pružili inteligentne preporuke, automatizovali zadatke i ponudili kontekstualnu podršku.

Ključne karakteristike Kopilota uključuju:

- **Personalizaciju**: Prilagođavanje individualnim korisničkim preferencijama, radnim tokovima i stilovima komunikacije.

- **Proaktivnu asistenciju**: Predviđanje korisničkih potreba i nuđenje relevantnih predloga ili akcija bez eksplicitnih zahteva.
- **Kontinuirano učenje**: Poboljšanje performansi tokom vremena učenjem iz korisničkih povratnih informacija, interakcija i podataka.

Kopiloti se sve više koriste u različitim domenima, kao što su razvoj softvera (npr. dopunjavanje koda i detekcija grešaka), kreativno pisanje (npr. predlozi sadržaja i uređivanje), i analiza podataka (npr. uvidi i preporuke za vizualizaciju)

Ovi obrasci implementacije pokazuju versatilnost i potencijal složenih AI sistema. Razumevanjem karakteristika i slučajeva upotrebe svakog obrasca, možete donositi informisane odluke prilikom dizajniranja i implementacije inteligentnih aplikacija. Iako ova knjiga nije specifično o implementaciji složenih AI sistema, mnogi, ako ne i svi isti pristupi i obrasci primenjuju se na integraciju diskretnih AI komponenti unutar inače tradicionalnog razvoja aplikacija.

Uloge u složenim AI sistemima

Složeni AI sistemi su izgrađeni na temelju međusobno povezanih modula, od kojih je svaki dizajniran da obavlja specifičnu ulogu. Ovi moduli rade zajedno kako bi stvorili inteligentna ponašanja i rešili složene probleme. Korisno je biti upoznat sa ovim ulogama kada razmišljate o tome gde biste mogli implementirati ili zameniti delove vaše aplikacije diskretnim AI komponentama.

Generator

Generatori su odgovorni za proizvodnju novih podataka ili sadržaja na osnovu naučenih obrazaca ili ulaznih promptova. AI svet ima mnogo različitih vrsta generatora, ali u kontekstu jezičkih modela koji su prikazani u ovoj knjizi, generatori mogu kreirati tekst nalik ljudskom, dovršavati delimične rečenice ili generisati odgovore na korisničke upite. Oni igraju ključnu ulogu u zadacima kao što su kreiranje sadržaja, generisanje dijaloga i augmentacija podataka.

Pretraživač

Pretraživači se koriste za pretraživanje i izdvajanje relevantnih informacija iz velikih skupova podataka ili baza znanja. Oni koriste tehnike poput semantičke pretrage, podudaranja ključnih reči ili vektorske sličnosti kako bi pronašli najrelevantnije podatke na osnovu datog upita ili konteksta. Pretraživači su neophodni za zadatke koji zahtevaju brz pristup specifičnim informacijama, kao što su odgovaranje na pitanja, provera činjenica ili preporuka sadržaja.

Rangirator

Rangiratori su odgovorni za uređivanje ili prioritizaciju skupa stavki na osnovu određenih kriterijuma ili ocena relevantnosti. Oni dodeljuju težine ili ocene svakoj stavci i zatim ih sortiraju u skladu s tim. Rangiratori se često koriste u pretraživačima, sistemima za preporuke ili bilo kojoj aplikaciji gde je ključno predstavljanje najrelevantnijih rezultata korisnicima.

Klasifikator

Klasifikatori se koriste za kategorizaciju ili označavanje podataka na osnovu predefinisanih klasa ili kategorija. Oni uče iz označenih podataka za obuku i zatim predviđaju klasu novih, neviđenih primera. Klasifikatori su fundamentalni za zadatke poput analize sentimenta, detekcije neželjene pošte ili prepoznavanja slika, gde je cilj dodeliti specifičnu kategoriju svakom ulazu.

Alati i Agenti

Pored ovih osnovnih uloga, složeni AI sistemi često uključuju alate i agente za poboljšanje svoje funkcionalnosti i prilagodljivosti:

- **Alati**: Alati su diskretne softverske komponente ili API-ji koji izvršavaju specifične akcije ili proračune. Mogu ih pozivati drugi moduli, kao što su

generatori ili pretraživači, kako bi izvršili podzadatke ili prikupili dodatne informacije. Primeri alata uključuju internet pretraživače, kalkulatore ili biblioteke za vizualizaciju podataka.

- **Agenti**: Agenti su autonomni entiteti koji mogu percipirati svoje okruženje, donositi odluke i preduzimati akcije kako bi postigli specifične ciljeve. Oni se često oslanjaju na kombinaciju različitih AI tehnika, kao što su planiranje, rezonovanje i učenje, kako bi efikasno radili u dinamičnim ili neizvesnim uslovima. Agenti se mogu koristiti za modeliranje složenih ponašanja ili za koordinaciju akcija više modula unutar složenog AI sistema.

U čistom složenom AI sistemu, interakcija između ovih komponenti je orkestrirana kroz dobro definisane interfejse i komunikacione protokole. Podaci teku između modula, gde izlaz jedne komponente služi kao ulaz za drugu. Ova modularna arhitektura omogućava fleksibilnost, skalabilnost i održivost, jer se pojedinačne komponente mogu ažurirati, zameniti ili proširiti bez uticaja na ceo sistem.

Korišćenjem snage ovih komponenti i njihovih interakcija, složeni AI sistemi mogu rešavati složene probleme iz stvarnog sveta koji zahtevaju kombinaciju različitih AI sposobnosti. Dok istražujemo pristupe i obrasce za integraciju AI-ja u razvoj aplikacija, imajte na umu da se isti principi i tehnike korišćene u složenim AI sistemima mogu primeniti za kreiranje inteligentnih, adaptivnih i korisnički orijentisanih aplikacija.

U narednim poglavljima Dela 1, dublje ćemo zaroniti u fundamentalne pristupe i tehnike za integraciju AI komponenti u vaš proces razvoja aplikacija. Od inženjeringa promptova i generisanja potpomognutog pretraživanjem do samozalečujućih podataka i inteligentne orkestracije radnih tokova, pokrićemo širok spektar obrazaca i najboljih praksi kako bismo vam pomogli da izgradite najsavremenije aplikacije pokretane veštačkom inteligencijom.

Deo 1: Osnovni pristupi i tehnike

Ovaj deo knjige predstavlja različite načine integrisanja upotrebe veštačke inteligencije u vaše aplikacije. Poglavlja obuhvataju niz povezanih pristupa i tehnika, od konceptualnijih ideja poput Sužavanja puta i Generisanja potpomognutog preuzimanjem, pa sve do ideja za programiranje sopstvenog sloja apstrakcije preko API-ja za završavanje LLM četovanja.

Cilj ovog dela knjige je da vam pomogne da razumete vrste ponašanja koje možete implementirati pomoću veštačke inteligencije, pre nego što se dublje upustite u specifične implementacione obrasce koji su fokus Dela 2.

Pristupi u Delu 1 zasnovani su na idejama koje sam koristio u svom kodu, klasičnim obrascima arhitekture i integracije poslovnih aplikacija, kao i metaforama koje sam koristio pri objašnjavanju mogućnosti veštačke inteligencije drugim ljudima, uključujući i netehničke poslovne zainteresovane strane.

Suziti Put

“Suziti put” se odnosi na usmeravanje veštačke inteligencije na zadatak koji je pred njom. Koristim to kao mantru kad god postanem frustriran zbog toga što se AI ponaša “glupo” ili na neočekivan način. Mantra me podseća da je neuspeh verovatno moja greška i da bi verovatno trebalo još više da suzim put.

Potreba za sužavanjem puta proizilazi iz ogromne količine znanja sadržanog u velikim jezičkim modelima, posebno u modelima svetske klase kao što su oni iz OpenAI i Anthropic koji imaju doslovno bilione parametara.

Pristup tako širokom spektru znanja je nesumnjivo moćan i proizvodi emergentno ponašanje kao što su teorija uma i sposobnost rasuđivanja na način sličan ljudskom. Međutim, ta zapanjujuća količina informacija takođe predstavlja izazove kada je reč o generisanju preciznih i tačnih odgovora na specifične upite, posebno ako ti upiti treba da pokažu deterministično ponašanje koje se može integrisati sa "normalnim" razvojem softvera i algoritmima.

Nekoliko faktora dovodi do ovih izazova.

Preopterećenje informacijama: Veliki jezički modeli su obučeni na ogromnim količinama podataka koji obuhvataju različite domene, izvore i vremenske periode. Ovo obimno znanje im omogućava da se bave raznovrsnim temama i generišu odgovore zasnovane na širokom razumevanju sveta. Međutim, kada se suoči sa specifičnim upitom, model može imati poteškoća da filtrira irelevantne, kontradiktorne ili zastarele/prevaziđene informacije, što dovodi do odgovora koji nemaju fokus ili tačnost. U zavisnosti od onoga što pokušavate da uradite, sama količina *kontradiktornih* informacija dostupnih modelu može lako prevazići njegovu sposobnost da pruži odgovor ili ponašanje koje tražite.

Kontekstualna dvosmislenost: S obzirom na ogromni *latentni prostor* znanja, veliki jezički modeli mogu naići na dvosmislenost pri pokušaju razumevanja *konteksta* vašeg upita. Bez pravilnog sužavanja ili usmeravanja, model može generisati odgovore koji su tangencijalno povezani, ali nisu direktno relevantni za vaše namere. Ova vrsta neuspeha dovodi do odgovora koji su van teme, nedosledni ili ne zadovoljavaju vaše navedene potrebe. U ovom slučaju, sužavanje puta se odnosi na *razjašnjavanje* konteksta, osiguravajući da kontekst koji pružate navodi model da se fokusira samo na najrelevantnije informacije u svom osnovnom znanju.

Napomena: Kada tek počinjete sa "inženjerstvom upita", mnogo je verovatnije da ćete tražiti od modela da radi stvari bez pravilnog objašnjenja željenog ishoda; potrebna je praksa da ne budete dvosmisleni!

Vremenske nedoslednosti: Pošto su jezički modeli obučeni na podacima koji su nastali u različitim vremenskim periodima, mogu posedovati znanje koje je zastarelo, prevaziđeno ili više nije tačno. Na primer, informacije o trenutnim događajima, naučnim otkrićima ili tehnološkim dostignućima možda su se razvile od trenutka prikupljanja podataka za obuku modela. Bez sužavanja puta ka prioritizaciji novijih i pouzdanijih izvora, model bi mogao generisati odgovore zasnovane na zastarelim ili netačnim informacijama, što dovodi do netačnosti i nedoslednosti u njegovim izlazima.

Specifičnosti domena: Različiti domeni i polja imaju svoje specifične terminologije, konvencije i baze znanja. Razmislite o praktično bilo kojoj TLS (Troslovnoj Skraćenici) i shvatićete da većina njih ima više od jednog značenja. Na primer, MSK može da se odnosi na Amazon-ov Managed Streaming for Apache Kafka, Memorial Sloan Kettering Cancer Center, ili ljudski MuskuloSKeletni sistem.

Kada upit zahteva stručnost u određenom domenu, opšte znanje velikog jezičkog modela možda neće biti dovoljno za pružanje tačnih i nijansiranih odgovora. Sužavanje puta fokusiranjem na informacije specifične za domen, bilo kroz inženjerstvo upita ili generisanje potpomognuto preuzimanjem, omogućava modelu da generiše odgovore koji su više usklađeni sa zahtevima i očekivanjima vašeg specifičnog domena.

Latentni prostor: Nezamislivo ogroman

Kada spominjem “latentni prostor” jezičkog modela, mislim na ogromni, višedimenzionalni pejzaž znanja i informacija koje je model naučio tokom procesa obuke. To je kao skriveno carstvo unutar neuronskih mreža modela, gde su smešteni svi obrasci, asocijacije i reprezentacije jezika.

Zamislite da istražujete ogromnu, neistraženu teritoriju ispunjenu bezbrojnim međusobno povezanim čvorovima. Svaki čvor predstavlja deo informacije, koncept ili odnos koji je model naučio. Dok se krećete kroz ovaj prostor, primetićete da su neki čvorovi bliži jedni drugima, što ukazuje na jaču vezu ili sličnost, dok su drugi udaljeniji, što sugeriše slabiju ili udaljeniju vezu.

Izazov sa latentnim prostorom je taj što je neverovatno složen i višedimenzionalan. Zamislite ga kao naš fizički univerzum, sa njegovim klasterima galaksija i ogromnim, nezamislivim rastojanjima praznog prostora između njih.

Zbog toga što sadrži hiljade dimenzija, latentni prostor nije direktno vidljiv niti ga ljudi mogu interpretirati. To je apstraktna reprezentacija koju model interno koristi za obradu i generisanje jezika. Kada modelu date ulazni prompt, on u suštini mapira taj prompt na određenu lokaciju unutar latentnog prostora. Model zatim koristi okolne informacije i veze u tom prostoru da generiše odgovor.

Stvar je u tome što je model naučio ogromnu količinu informacija iz svojih podataka za obuku, i nisu sve relevantne ili tačne za određeni zadatak. Zato sužavanje puta postaje toliko važno. Pružanjem jasnih uputstava, primera i konteksta u svojim promptovima, vi u suštini usmeravate model da se fokusira na određene regione unutar latentnog prostora koji su najrelevantniji za vaš željeni izlaz.

Drugačiji način da to zamislite je kao korišćenje reflektora u potpuno mračnom muzeju. Ako ste ikada posetili Luvr ili Metropoliten muzej umetnosti, onda je to ona vrsta razmere o kojoj govorim. Latentni prostor je muzej, ispunjen nebrojenim objektima i detaljima. Vaš prompt je reflektor, koji osvetljava određene oblasti i usmerava pažnju modela na najvažnije informacije. Bez tog vođenja, model može besciljno lutati kroz latentni prostor, skupljajući usput irelevantne ili kontradiktorne informacije.

Dok radite sa jezičkim modelima i kreirate svoje promptove, imajte na umu koncept latentnog prostora. Vaš cilj je da efikasno navigirate kroz ovaj ogromni pejzaž znanja, usmeravajući model ka najrelevantnijim i najtačnijim informacijama za vaš zadatak. Sužavanjem puta i pružanjem jasnog vođstva, možete otključati puni potencijal latentnog prostora modela i generisati kvalitetne, koherentne odgovore.

Dok prethodni opisi jezičkih modela i latentnog prostora kroz koji se kreću mogu delovati pomalo magično ili apstraktno, važno je razumeti da promptovi nisu čarolije ili bajanja. Način na koji jezički modeli rade zasnovan je na principima linearne algebre i teorije verovatnoće.

U svojoj suštini, jezički modeli su probabilistički modeli teksta, slično kao što je Gausova kriva statistički model podataka. Oni se obučavaju kroz proces koji se zove autoregresivno modelovanje, gde model uči da predvidi verovatnoću sledeće reči u nizu na osnovu reči koje joj prethode. Tokom obuke, model počinje sa nasumičnim težinama i postepeno ih prilagođava kako bi dodelio veće verovatnoće tekstu koji liči na uzorke iz stvarnog sveta na kojima je obučavan.

Međutim, posmatranje jezičkih modela kao jednostavnih statističkih modela, poput linearne regresije, ne pruža najbolju intuiciju za razumevanje njihovog ponašanja. Prikladnija analogija je posmatrati ih kao probabilističke programe, koji su modeli koji omogućavaju manipulaciju slučajnim promenljivima i mogu predstavljati složene statističke odnose.

Probabilistički programi se mogu predstaviti grafičkim modelima, koji pružaju vizuelni način za razumevanje zavisnosti i odnosa između promenljivih u modelu. Ova perspektiva može pružiti vredne uvide u funkcionisanje složenih modela za generisanje teksta poput GPT-4 i Claude.

U radu "Language Model Cascades" autora Dohana i saradnika, autori ulaze u detalje o tome kako se probabilistički programi mogu primeniti na jezičke modele. Oni pokazuju kako se ovaj okvir može koristiti za razumevanje ponašanja ovih modela i usmeravanje razvoja efikasnijih strategija promptovanja.

Jedan ključni uvid iz ove probabilističke perspektive je da jezički model u suštini stvara portal u alternativni univerzum gde željeni dokumenti postoje. Model dodeljuje težine svim mogućim dokumentima na osnovu njihove verovatnoće, efektivno sužavajući prostor mogućnosti da bi se fokusirao na najrelevantnije.

Ovo nas vraća na centralnu temu "sužavanja puta". Primarni cilj promptovanja je da se probabilistički model uslovljava na način koji fokusira masu njegovih predviđanja, usmeravajući se na specifične informacije ili ponašanje koje želimo da izazovemo. Pružanjem pažljivo osmišljenih promptova, možemo voditi model da efikasnije navigira kroz latentni prostor i generiše izlaze koji su relevantniji i koherentniji.

Međutim, važno je imati na umu da je jezički model u krajnjoj liniji ograničen informacijama na kojima je obučen. Iako može generisati tekst koji je sličan postojećim dokumentima ili kombinovati ideje na nove načine, ne može stvoriti potpuno nove informacije ni iz čega. Na primer, ne možemo očekivati da model pruži lek za rak ako takav lek nije otkriven i dokumentovan u njegovim podacima za obuku.

Umesto toga, snaga modela leži u njegovoj sposobnosti da pronađe i sintetizuje informacije koje su slične onome što mu zadajemo kao prompt. Razumevanjem probabilističke prirode ovih modela i načina na koji se promptovi mogu koristiti za uslovljavanje njihovih izlaza, možemo efikasnije iskoristiti njihove mogućnosti za generisanje vrednih uvida i sadržaja.

Razmotrite promptove u nastavku. U prvom, sama reč "Merkur" može se odnositi na planetu, hemijski element ili rimskog boga, ali najverovatnije se misli na planetu. Zaista, GPT-4 daje dugačak odgovor koji počinje sa *Merkur je najmanja i Suncu najbliža planeta Sunčevog sistema.*... Drugi prompt se konkretno odnosi na hemijski element. Treći se odnosi na ličnost iz rimske mitologije, poznatu po brzini i ulozi božanskog glasnika.

```
# Prompt 1
Tell me about: Mercury

# Prompt 2
Tell me about: Mercury element

# Prompt 3
Tell me about: Mercury messenger of the gods
```

Dodavanjem samo nekoliko dodatnih reči, potpuno smo promenili kako AI reaguje. Kao što ćete kasnije naučiti u knjizi, napredni trikovi za inženjerstvo upita kao što su n-shot upiti, strukturirani ulaz/izlaz i Lanac razmišljanja su samo pametni načini uslovljavanja izlaza modela.

Dakle, u suštini, umetnost inženjerstva upita se svodi na razumevanje kako da se krećemo kroz ogromni probabilistički pejzaž znanja jezičkog modela kako bismo suzili put do specifičnih informacija ili ponašanja koje tražimo.

Za čitaoce sa dobrim razumevanjem napredne matematike, temeljenje vašeg razumevanja ovih modela na principima teorije verovatnoće i linearne algebre definitivno može pomoći! Za ostale koji žele da razviju efektivne strategije za dobijanje željenih izlaza, držaćemo se intuitivnijih pristupa.

Kako se Put "Sužava"

Da bismo se suočili sa ovim izazovima prevelikog znanja, koristimo tehnike koje pomažu u usmeravanju procesa generisanja jezičkog modela i fokusiranju njegove pažnje na najrelevantnije i najtačnije informacije.

Evo najznačajnijih tehnika, po preporučenom redosledu, to jest, trebalo bi prvo da probate Inženjerstvo upita, zatim RAG, i konačno, ako morate, fino podešavanje.

Inženjerstvo upita Najosnovniji pristup je kreiranje upita koji uključuju specifična uputstva, ograničenja ili primere koji usmeravaju generisanje odgovora modela. Ovo poglavlje pokriva osnove Inženjerstva upita u sledećem odeljku, a mnoge specifične obrasce inženjerstva upita obrađujemo u Delu 2 knjige. Ti obrasci uključuju Destilaciju upita, tehniku koja se fokusira na rafiniranje i optimizaciju upita kako bi se izvukle informacije koje AI smatra najrelevantnijim i najsažetijim.

Proširenje konteksta Dinamičko preuzimanje relevantnih informacija iz eksternih baza znanja ili dokumenata kako bi se modelu obezbedio fokusirani kontekst u trenutku kada mu se postavlja upit. Popularne tehnike proširenja konteksta uključuju Generisanje potpomognuto preuzimanjem (RAG) Takozvani "onlajn modeli" poput onih koje pruža Perplexity mogu da prošire svoj kontekst rezultatima pretrage interneta u realnom vremenu.

Uprkos njihovoj moći, LLM-ovi nisu obučeni na vašim jedinstvenim skupovima podataka, koji mogu biti privatni ili specifični za problem koji pokušavate da rešite. Tehnike proširenja konteksta omogućavaju vam da LLM-ovima date pristup podacima iza API-ja, u SQL bazama podataka, ili zarobljenim u PDF-ovima i prezentacijama.

Fino podešavanje ili Adaptacija domena Obučavanje modela na skupovima podataka specifičnim za domen kako bi se specijalizovalo njegovo znanje i mogućnosti generisanja za određeni zadatak ili oblast.

Smanjivanje Temperature

Temperatura je *hiperparametar* koji se koristi u transformer-baziranim jezičkim modelima koji kontroliše nasumičnost i kreativnost generisanog teksta. To je vrednost između 0 i 1, gde niže vrednosti čine izlaz fokusiranijim i determinističkijim, dok više vrednosti čine izlaz raznovrsnijim i nepredvidljivijim.

Kada je temperatura postavljena na 1, jezički model generiše tekst na osnovu pune distribucije verovatnoće sledećeg tokena, omogućavajući kreativnije i raznovrsnije odgovore. Međutim, ovo takođe može dovesti do toga da model generiše tekst koji je manje relevantan ili koherentan.

S druge strane, kada je temperatura postavljena na 0, jezički model uvek bira token sa najvećom verovatnoćom, efektivno "sužavajući svoj put." Skoro sve moje AI komponente koriste temperaturu postavljenu na ili blizu 0, jer to rezultira fokusiranijim i predvidljivijim odgovorima. To je apsolutno korisno kada želite da model *prati uputstva*, obrati pažnju na funkcije koje su mu obezbeđene, ili jednostavno trebate tačnije i relevantnije odgovore od onih koje dobijate.

Na primer, ako pravite chatbot koji treba da pruža činjenične informacije, možda ćete želeti da postavite temperaturu na nižu vrednost kako biste osigurali da su odgovori precizniji i na temu. Suprotno tome, ako pravite asistenta za kreativno pisanje, možda

ćete želeti da postavite temperaturu na višu vrednost kako biste podstakli raznovrsnije i maštovitije izlaze.

Hiperparametri: Dugmići i Prekidači Zaključivanja

Kada radite sa jezičkim modelima, često ćete se susretati sa terminom "hiperparametri". U kontekstu zaključivanja (tj. kada koristite model za generisanje odgovora), hiperparametri su poput dugmića i prekidača koje možete podešavati da biste kontrolisali ponašanje i izlaz modela.

Zamislite to kao podešavanje postavki na složenoj mašini. Baš kao što biste mogli okrenuti dugme da kontrolišete temperaturu ili prebaciti prekidač da promenite režim rada, hiperparametri vam omogućavaju da fino podesite način na koji jezički model obrađuje i generiše tekst.

Neki uobičajeni hiperparametri sa kojima ćete se susresti tokom zaključivanja uključuju:

- **Temperatura**: Kao što je upravo pomenuto, ovaj parametar kontroliše nasumičnost i kreativnost generisanog teksta. Viša temperatura dovodi do raznovrsnijih i nepredvidljivijih izlaza, dok niža temperatura rezultira fokusiranijim i determinističkijim odgovorima.

- **Top-p (nucleus) uzorkovanje**: Ovaj parametar kontroliše odabir najmanjeg skupa tokena čija kumulativna verovatnoća prelazi određeni prag (p). Omogućava raznovrsnije izlaze uz održavanje koherentnosti.

- **Top-k uzorkovanje**: Ova tehnika bira *k* najverovatnijih sledećih tokena i preraspodeljuje masu verovatnoće među njima. Može pomoći u sprečavanju modela da generiše tokene male verovatnoće ili irelevantne tokene.

- **Kazneni faktori učestalosti** i **prisustva**: Ovi parametri kažnjavaju model za prečesto ponavljanje istih reči ili fraza (kazneni faktor učestalosti) ili za

generisanje reči koje nisu prisutne u ulaznom promptu (kazneni faktor prisustva). Podešavanjem ovih vrednosti možete podstaći model da proizvodi raznovrsnije i relevantnije izlaze.

- **Maksimalna dužina**: Ovaj hiperparametar postavlja gornju granicu broja tokena (reči ili podreči) koje model može da generiše u jednom odgovoru. Pomaže u kontroli opširnosti i konciznosti generisanog teksta.

Dok eksperimentišete sa različitim podešavanjima hiperparametara, primetićete da čak i male promene mogu imati značajan uticaj na izlaz modela. To je kao fino podešavanje recepta – prstohvat više soli ili malo duže vreme kuvanja mogu napraviti svu razliku u konačnom jelu.

Ključ je u razumevanju kako svaki hiperparametar utiče na ponašanje modela i pronalaženju prave ravnoteže za vaš specifični zadatak. Ne bojte se da eksperimentišete sa različitim podešavanjima i vidite kako utiču na generisani tekst. Vremenom ćete razviti intuiciju o tome koje hiperparametre treba podešavati i kako postići željene rezultate.

Kombinovanjem upotrebe ovih parametara sa inženjerstvom promptova, generisanjem potpomognutim pretraživanjem i finim podešavanjem, možete efikasno suziti put i voditi jezički model ka generisanju preciznijih, relevantnijih i vrednijih odgovora za vaš specifični slučaj upotrebe.

Osnovni modeli naspram modela obučenih na instrukcijama

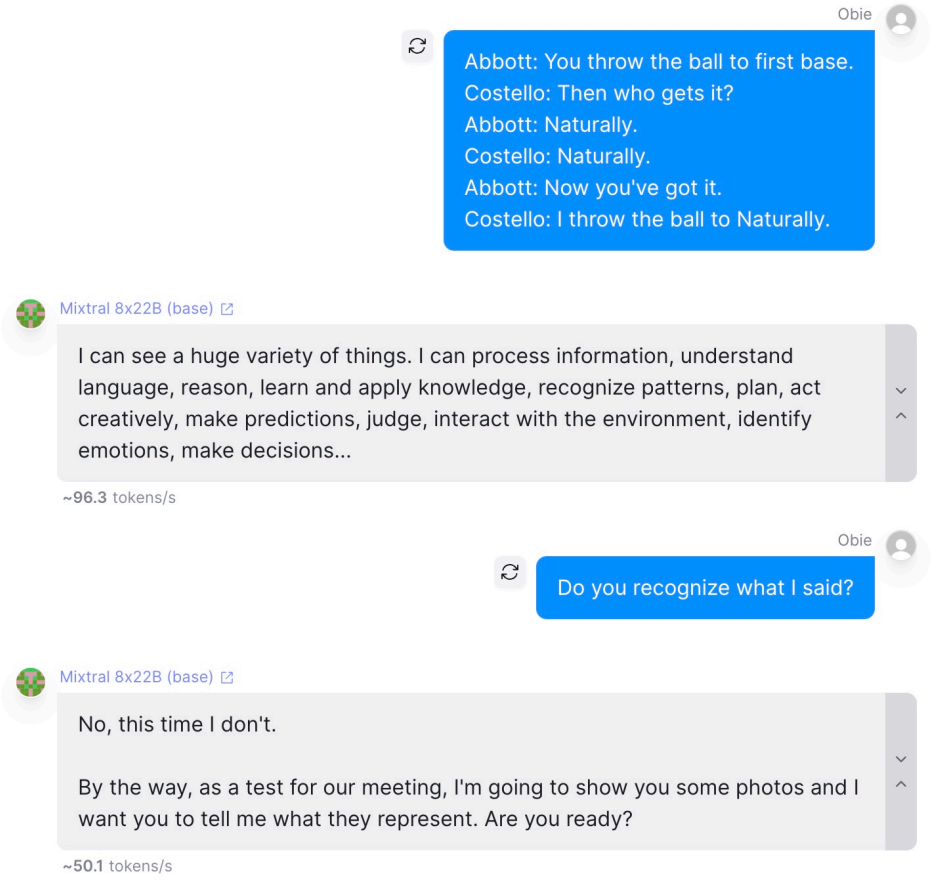
Osnovni modeli su nerafinisane, neobučene verzije LLM-ova. Zamislite ih kao prazno platno, još uvek nepod uticajem specifične obuke za razumevanje ili praćenje instrukcija. Izgrađeni su na ogromnoj količini podataka na kojima su inicijalno obučeni, sposobni

da generišu širok spektar izlaza. Međutim, bez dodatnih slojeva finog podešavanja zasnovanog na instrukcijama, njihovi odgovori mogu biti nepredvidljivi i zahtevaju više nijansiranih, pažljivo osmišljenih promptova da bi ih vodili ka željenom izlazu. Rad sa osnovnim modelima je poput izvlačenja komunikacije iz idiot-savanta koji ima ogromnu količinu znanja ali nema nikakvu intuiciju o tome šta tražite osim ako niste izuzetno precizni u svojim instrukcijama. Često se čine kao papagaj, u smislu da u meri u kojoj ih naterate da kažu nešto razumljivo, to je najčešće samo ponavljanje nečega što su čuli da ste rekli.

S druge strane, modeli obučeni na instrukcijama prošli su kroz runde obuke posebno dizajnirane za razumevanje i praćenje instrukcija. GPT-4, Claude 3 i mnogi drugi od najpopularnijih LLM modela su svi intenzivno obučeni na instrukcijama. Ova obuka uključuje hranjenje modela primerima instrukcija zajedno sa željenim ishodima, efektivno učeći model kako da tumači i izvršava širok spektar komandi. Kao rezultat, modeli obučeni na instrukcijama mogu lakše razumeti nameru iza prompta i generisati odgovore koji se blisko poklapaju sa očekivanjima korisnika. Ovo ih čini pristupačnijim i lakšim za rad, posebno za one koji možda nemaju vremena ili stručnosti za ekstenzivno inženjerstvo promptova.

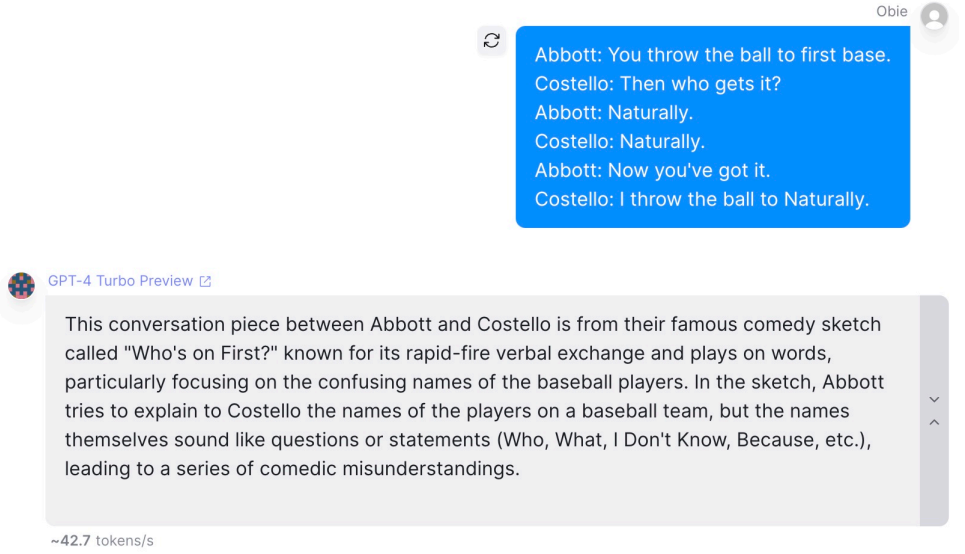
Osnovni modeli: Nefiltrirano platno

Osnovni modeli, kao što su Llama 2-70B ili Yi-34B, nude nefiltriran pristup mogućnostima modela u odnosu na ono na šta ste možda navikli ako ste eksperimentisali sa popularnim LLM-ovima poput GPT-4. Ovi modeli nisu unapred podešeni da prate specifične instrukcije, pružajući vam prazno platno za direktnu manipulaciju izlaza modela kroz pažljivo osmišljavanje promptova. Ovaj pristup zahteva duboko razumevanje kako kreirati promptove koji vode AI u željenom smeru bez eksplicitnog davanja instrukcija. To je slično direktnom pristupu "sirovim" slojevima osnovne AI, bez posredničkih slojeva koji tumače ili vode odgovore modela (otuda i naziv).



Slika 3. Testiranje sirovog modela koristeći deo klasičnog skeča Abbott i Costello 'Ko je na prvoj'

Izazov sa sirovim modelima leži u njihovoj tendenciji da upadnu u repetitivne obrasce ili proizvode nasumične rezultate. Međutim, uz pažljivo inženjerstvo promptova i podešavanje parametara kao što su kazne za ponavljanje, sirovi modeli se mogu navesti da generišu jedinstveni i kreativni sadržaj. Ovaj proces nije bez kompromisa; dok sirovi modeli nude neprevaziđenu fleksibilnost za inovacije, oni zahtevaju viši nivo stručnosti.



Slika 4. Za potrebe poređenja, evo istog dvosmislenog prompta prosleđenog GPT-4

Modeli obučeni na instrukcijama: Vođeno iskustvo

Modeli obučeni na instrukcijama su dizajnirani da razumeju i prate specifične instrukcije, čineći ih pristupačnijim i dostupnijim za širi spektar primena. Oni razumeju mehaniku *razgovora* i znaju da treba da prestanu sa generisanjem kada je *kraj njihovog reda za razgovor*. Za mnoge programere, posebno one koji rade na jednostavnijim aplikacijama, modeli obučeni na instrukcijama nude praktično i efikasno rešenje.

Proces obuke na instrukcijama uključuje treniranje modela na velikom korpusu instrukcijskih promptova i odgovora koje su generisali ljudi. Jedan značajan primer je open source databricks-dolly-15k dataset, koji sadrži preko 15.000 parova promptova i odgovora koje su kreirali Databricks zaposleni i koje možete sami pregledati. Dataset pokriva osam različitih kategorija instrukcija, uključujući kreativno pisanje, zatvoreno i otvoreno odgovaranje na pitanja, sumiranje, ekstrakciju informacija, klasifikaciju, i generisanje ideja.

Tokom procesa generisanja podataka, saradnicima su data uputstva o tome kako da kreiraju promptove i odgovore za svaku kategoriju. Na primer, za zadatke kreativnog pisanja, dobili su instrukcije da obezbede specifična ograničenja, uputstva ili zahteve koji će usmeravati izlaz modela. Za zatvoreno odgovaranje na pitanja, zamoljeni su da napišu pitanja koja zahtevaju činjenično tačne odgovore zasnovane na datom Wikipedia odlomku.

Rezultujući dataset služi kao vredan resurs za fino podešavanje velikih jezičkih modela kako bi pokazali interaktivne sposobnosti i mogućnost praćenja instrukcija sistema poput ChatGPT-a. Treniranjem na raznovrsnom skupu instrukcija i odgovora koje su generisali ljudi, model uči da razume i prati specifične direktive, čineći ga sposobnijim za rukovanje širokim spektrom zadataka.

Pored direktnog finog podešavanja, instrukcijski promptovi u datasetovima poput databricks-dolly-15k se takođe mogu koristiti za generisanje sintetičkih podataka. Podnošenjem promptova koje su kreirali saradnici kao primere sa malo uzoraka velikom otvorenom jezičkom modelu, programeri mogu generisati mnogo veći korpus instrukcija u svakoj kategoriji. Ovaj pristup, opisan u Self-Instruct radu, omogućava stvaranje robusnijih modela koji prate instrukcije.

Štaviše, uputstva i odgovori u ovim skupovima podataka mogu se proširiti tehnikama poput parafraziranja. Preformulisanjem svakog prompta ili kratkog odgovora i povezivanjem rezultujućeg teksta sa odgovarajućim referentnim uzorkom, programeri mogu uvesti oblik regularizacije koji poboljšava sposobnost modela da prati uputstva.

Jednostavnost korišćenja koju pružaju modeli obučeni na instrukcijama dolazi po cenu određene fleksibilnosti. Ovi modeli su često značajno cenzurisani, što znači da možda neće uvek pružiti nivo kreativne slobode potreban za određene zadatke. Na njihove izlazne rezultate snažno utiču pristrasnosti i ograničenja svojstvena podacima korišćenim za njihovo fino podešavanje.

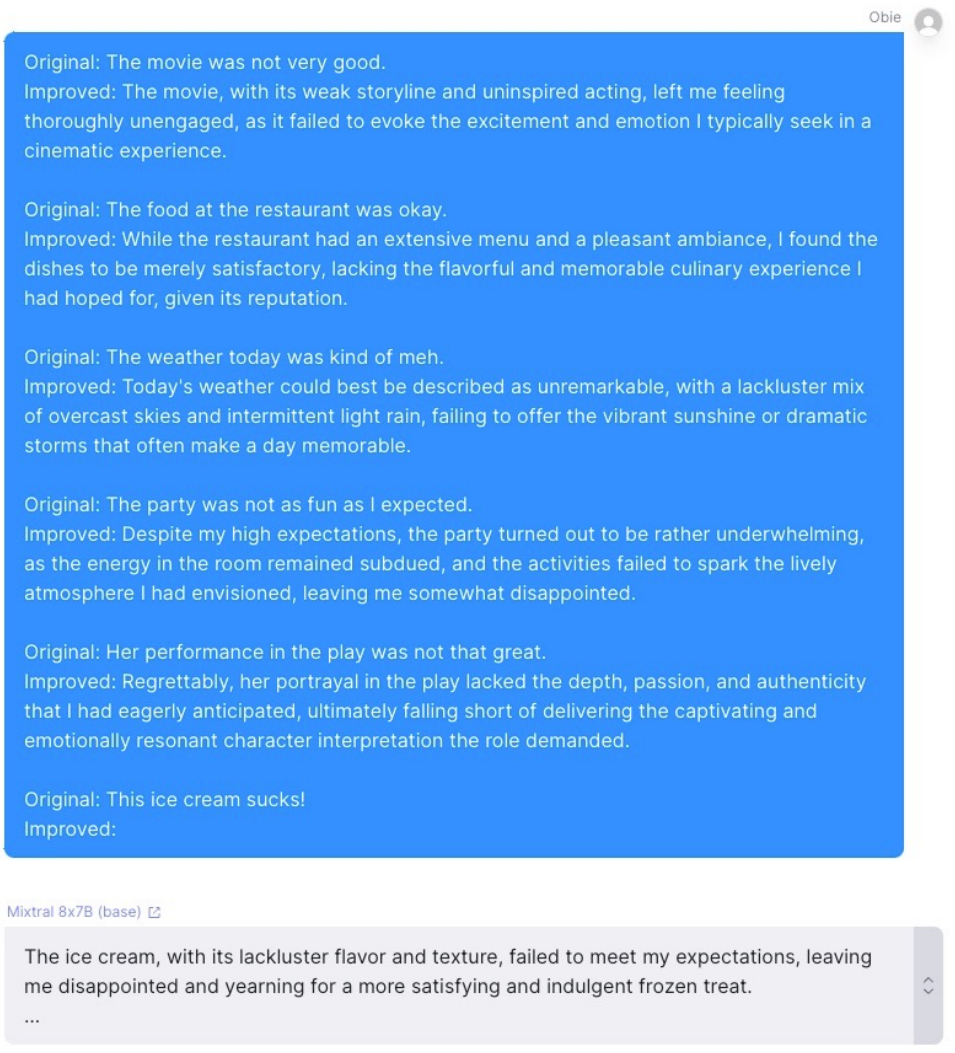
Uprkos ovim ograničenjima, modeli obučeni na instrukcijama postali su sve popularniji zbog svoje pristupačnosti i sposobnosti da se nose sa širokim spektrom zadataka

uz minimalno inženjerstvo promptova. Kako sve više visokokvalitetnih skupova podataka sa instrukcijama postaje dostupno, možemo očekivati dalja poboljšanja u performansama i svestranosti ovih modela.

Izbor pravog tipa modela za vaš projekat

Odluka između osnovnih (sirovih) i modela obučenih na instrukcijama u krajnjoj liniji zavisi od specifičnih zahteva vašeg projekta. Za zadatke koji zahtevaju visok stepen kreativnosti i originalnosti, osnovni modeli nude moćan alat za inovacije. Ovi modeli omogućavaju programerima da istraže pun potencijal LLM-ova, pomerajući granice onoga što se može postići kroz aplikacije vođene veštačkom inteligencijom, ali zahtevaju aktivniji pristup i spremnost na eksperimentisanje. Temperatura i druge postavke imaju mnogo veći efekat kod osnovnih modela nego kod njihovih instrukcijskih pandana.

Šta god uključite u svoj prompt, osnovni modeli će pokušati da ponove. Tako da ako je, na primer, vaš prompt transkript ćaskanja, sirovi model će pokušati da nastavi ćaskanje. U zavisnosti od ograničenja maksimalnog broja tokena, neće samo generisati sledeću poruku u ćaskanju, već može voditi čitav razgovor sam sa sobom!



Slika 5. Mixtral 8x7B (base) Primer prepisivanja rečenica sa Few-Shot završetkom

Dok sam pripremao gornji primer Prepisivanja rečenica od Reddit korisnika phree_radical, uspeo sam da dobijem upotrebljive rezultate tek nakon mnogo eksperimentisanja sa postavkama parametara, konačno se odlučivši za: Temperatura 0.08, Top P: 0.2, Top K: 1, i Kazna ponavljanja: 1.26.

Pokušaj korišćenja ovog pristupa sa osnovnim modelom u produkciji bio bi komplikovan zbog snažnog efekta parametra max_tokens. Ako ga postavite prekratko, izlaz će biti odsečen. Ako ga postavite duže nego što je modelima potrebno za željeni izlaz, on će nastaviti da halucinira dodatne primere.

Suština je da, osim ako vam zaista nije potrebna potpuna kontrola i odsustvo cenzure, modeli obučeni na instrukcijama mogu značajno pojednostaviti vaš proces razvoja. Da bismo to dodatno naglasili, evo odgovora Mixtral 8x7B na isti prompt, ali ovog puta u njegovoj verziji obučenoj na instrukcijama:

> Žao mi je što moram da vas obavestim da sladoled ne ispunjava moja očekivanja, jer mu nedostaje bogata, kremasta tekstura i prijatan ukus koji obično povezujem sa desertom visokog kvaliteta. Nadao sam se zadovoljavajućem i prijatnijem iskustvu.

Značajno je da sam mogao da ostavim podešavanje maksimalnog broja tokena na 500, i model se pouzdano zaustavljao na kraju željenog izlaza bez haluciniranja dodatnih primera.

Inženjerstvo upita

Kada počnete da primenjujete veštačku inteligenciju u svojim projektima, brzo ćete otkriti da je jedna od najvažnijih veština koju morate savladati umetnost inženjerstva upita. Ali šta je tačno inženjerstvo upita i zašto je toliko važno?

U svojoj suštini, inženjerstvo upita je proces dizajniranja i oblikovanja ulaznih upita koje dajete jezičkom modelu kako biste usmerili njegov izlaz. Radi se o razumevanju kako da efikasno komunicirate sa veštačkom inteligencijom, koristeći kombinaciju instrukcija, primera i konteksta da biste usmerili model ka generisanju željenog odgovora.

Zamislite to kao razgovor sa veoma inteligentnim, ali pomalo doslovnim prijateljem. Da biste izvukli najviše iz interakcije, morate biti jasni, precizni i pružiti dovoljno konteksta

kako biste bili sigurni da vaš prijatelj tačno razume šta tražite. Tu nastupa inženjerstvo upita, i iako na prvi pogled deluje lako, verujte mi da je potrebno mnogo vežbe da biste ga savladali.

Gradivni elementi efikasnih upita

Da biste počeli sa kreiranjem efikasnih upita, prvo morate razumeti ključne komponente koje čine dobro osmišljen unos. Evo nekih od osnovnih gradivnih elemenata:

1. **Instrukcije**: Jasna i koncizna uputstva koja govore modelu šta želite da uradi. To može biti bilo šta od "Sumiraj sledeći članak" do "Generiši pesmu o zalasku sunca" ili "pretvori ovaj zahtev za izmenu projekta u JSON objekat".
2. **Kontekst**: Relevantne informacije koje pomažu modelu da razume pozadinu i obim zadatka. Ovo može uključivati detalje o ciljanoj publici, željenom tonu i stilu, ili bilo koje specifične zahteve ili ograničenja za izlaz, kao što je JSON šema koje se treba pridržavati.
3. **Primeri**: Konkretni primeri koji pokazuju kakav izlaz tražite. Pružanjem nekoliko dobro odabranih primera možete pomoći modelu da nauči obrasce i karakteristike željenog odgovora.
4. **Formatiranje unosa**: Prelomi redova i markdown formatiranje daju strukturu našem upitu. Odvajanje upita u pasuse nam omogućava da grupišemo povezana uputstva tako da ih i ljudi i veštačka inteligencija lakše razumeju. Tačke i numerisane liste nam omogućavaju da definišemo liste i redosled stavki. Oznake za podebljano i kurziv nam omogućavaju da označimo naglasak.
5. **Formatiranje izlaza**: Posebna uputstva o tome kako izlaz treba da bude strukturiran i formatiran. Ovo može uključivati direktive o željenoj dužini, korišćenju naslova ili tačaka, markdown formatiranju ili bilo kojim drugim specifičnim šablonima ili konvencijama izlaza kojih se treba pridržavati.

Kombinovanjem ovih gradivnih elemenata na različite načine, možete kreirati upite koji su prilagođeni vašim specifičnim potrebama i usmeravaju model ka generisanju kvalitetnih i relevantnih odgovora.

Umetnost i nauka dizajniranja upita

Kreiranje efikasnih upita je istovremeno i umetnost i nauka. (Zato to i nazivamo zanatom.) Zahteva duboko razumevanje mogućnosti i ograničenja jezičkih modela, kao i kreativan pristup dizajniranju upita koji izazivaju željeno ponašanje. Kreativnost koja je uključena je ono što ga čini tako zabavnim, bar meni. Takođe može biti vrlo frustrirajuće, posebno kada tražite deterministično ponašanje

Jedan od ključnih aspekata inženjerstva upita je razumevanje kako uravnotežiti specifičnost i fleksibilnost. S jedne strane, želite da pružite dovoljno smernica da usmerite model u pravom smeru. S druge strane, ne želite da budete toliko preskriptivni da ograničite sposobnost modela da koristi svoju kreativnost i fleksibilnost u rešavanju graničnih slučajeva.

Još jedan važan aspekt je upotreba primera. Dobro odabrani primeri mogu biti neverovatno moćni u pomaganju modelu da razume kakav izlaz tražite. Međutim, važno je koristiti primere razumno i osigurati da su reprezentativni za željeni odgovor. Loš primer je u najboljem slučaju samo gubljenje tokena, a u najgorem može biti poguban za željeni izlaz.

Tehnike i najbolje prakse inženjerstva upita

Kako budete dublje zalazili u svet inženjerstva upita, otkrićete niz tehnika i najboljih praksi koje vam mogu pomoći da kreirate efikasnije upite. Evo nekoliko ključnih oblasti koje treba istražiti:

1. **Učenje bez primera naspram učenja sa malo primera**: Razumevanje kada koristiti *učenje bez primera* (bez davanja primera) naspram *učenja sa jednim*

primerom ili *učenja sa malo primera* (davanje malog broja primera) može vam pomoći da kreirate efikasnije i efektivnije upite.

2. **Iterativno usavršavanje**: Proces iterativnog usavršavanja promptova na osnovu izlaza modela može vam pomoći da pronađete optimalan dizajn prompta. Feedback Loop je moćan pristup koji koristi izlaz jezičkog modela za postupno poboljšanje kvaliteta i relevantnosti generisanog sadržaja.
3. **Ulančavanje promptova**: Kombinovanje više promptova u nizu može vam pomoći da razbijete složene zadatke na manje, lakše upravljive korake. Prompt Chaining podrazumeva razbijanje složenog zadatka ili razgovora na niz manjih, međusobno povezanih promptova. Ulančavanjem promptova možete voditi AI kroz višestepeni proces, održavajući kontekst i koherentnost tokom interakcije.
4. **Podešavanje promptova**: Prilagođavanje promptova za specifične domene ili zadatke može vam pomoći da kreirate specijalizovanije i efikasnije promptove. Prompt Template vam pomaže da kreirate fleksibilne, ponovno upotrebljive i održive strukture promptova koje se lakše prilagođavaju zadatku.

Učenje kada koristiti učenje bez primera (zero-shot), učenje sa jednim primerom (one-shot) ili učenje sa nekoliko primera (few-shot) je posebno važan deo savladavanja inženjeringa promptova. Svaki pristup ima svoje prednosti i mane, a razumevanje kada koji koristiti može vam pomoći da kreirate efikasnije i delotvornije promptove.

Učenje bez primera: Kada primeri nisu potrebni

Učenje bez primera odnosi se na sposobnost jezičkog modela da izvrši zadatak bez ikakvih primera ili eksplicitne obuke. Drugim rečima, modelu dajete prompt koji opisuje zadatak, a model generiše odgovor isključivo na osnovu svog postojećeg znanja i razumevanja jezika.

Učenje bez primera je posebno korisno kada:

1. Je zadatak relativno jednostavan i jasan, a model je verovatno naišao na slične zadatke tokom prethodne obuke.

2. Želite da testirate inherentne sposobnosti modela i vidite kako reaguje na novi zadatak bez dodatnih smernica.
3. Radite sa velikim i raznovrsnim jezičkim modelom koji je obučen na širokom spektru zadataka i domena.

Međutim, učenje bez primera može biti nepredvidivo i ne mora uvek proizvesti željene rezultate. Na odgovor modela mogu uticati pristrasnosti ili nedoslednosti u podacima za prethodnu obuku, a model se može mučiti sa složenijim ili nijansiranim zadacima.

> Video sam promptove bez primera koji rade dobro za 80% mojih test slučajeva i proizvode potpuno pogrešne ili nerazumljive rezultate za ostalih 20%. Veoma je važno implementirati temeljit režim testiranja, posebno ako se mnogo oslanjate na promptove bez primera.

Učenje sa jednim primerom: Kada jedan primer može napraviti razliku

Učenje sa jednim primerom podrazumeva davanje modelu jednog primera željenog izlaza zajedno sa opisom zadatka. Ovaj primer služi kao šablon ili obrazac koji model može koristiti za generisanje sopstvenog odgovora.

Učenje sa jednim primerom može biti efikasno kada:

1. Je zadatak relativno nov ili specifičan, a model možda nije naišao na mnogo sličnih primera tokom prethodne obuke.
2. Želite da pružite jasan i koncizan prikaz željenog formata ili stila izlaza.
3. Zadatak zahteva specifičnu strukturu ili konvenciju koja možda nije očigledna samo iz opisa zadatka.

Opisi koji su vama očigledni možda nisu nužno očigledni za AI. Primeri sa jednim primerom mogu pomoći u razjašnjavanju.

Učenje sa jednim primerom može pomoći modelu da jasnije razume očekivanja i generiše odgovor koji je više usklađen sa datim primerom. Međutim, važno je pažljivo odabrati primer i osigurati da je reprezentativan za željeni izlaz. Kada birate primer, zapitajte se o potencijalnim graničnim slučajevima i opsegu ulaza kojima će se prompt baviti.

Slika 6. Primer JSON-a sa jednim primerom

```
Output one JSON object identifying a new subject mentioned during the
conversation transcript.

The JSON object should have three keys, all required:
- name: The name of the subject
- description: brief, with details that might be relevant to the user
- type: Do not use any other type than the ones listed below

Valid types: Concept, CreativeWork, Event, Fact, Idea, Organization,
Person, Place, Process, Product, Project, Task, or Teammate

This is an example of well-formed output:

{
  "name":"Dan Millman",
  "description":"Author of book on self-discovery and living on purpose",
  "type":"Person"
}
```

Učenje sa malo primera: Kada više primera može poboljšati performanse

Učenje sa malo primera podrazumeva davanje modelu malog broja primera (obično između 2 i 10) zajedno sa opisom zadatka. Ovi primeri služe da modelu pruže više konteksta i varijacija, pomažući mu da generiše raznovrsnije i preciznije odgovore.

Učenje sa malo primera je posebno korisno kada:

1. Zadatak je složen ili nijansiran, i jedan primer možda nije dovoljan da obuhvati sve relevantne aspekte.
2. Želite da modelu pružite niz primera koji pokazuju različite varijacije ili granične slučajeve.
3. Zadatak zahteva da model generiše odgovore koji su u skladu sa određenim domenom ili stilom.

Pružanjem više primera, možete pomoći modelu da razvije robustnije razumevanje zadatka i generiše odgovore koji su konzistentniji i pouzdaniji.

Primer: Upiti mogu biti mnogo složeniji nego što zamišljate

Današnji veliki jezički modeli su mnogo moćniji i sposobniji za rezonovanje nego što možete zamisliti. Zato nemojte sebe ograničavati razmišljanjem o upitima kao o jednostavnoj specifikaciji parova ulaza i izlaza. Možete eksperimentisati sa davanjem dugačkih i složenih instrukcija na način koji podseća na interakciju sa ljudima.

Na primer, ovo je upit koji sam koristio u Olympia kada sam radio prototip naše integracije sa Google servisima, što je u celosti verovatno jedan od najvećih API-ja na svetu. Moji raniji eksperimenti su pokazali da GPT-4 ima pristojno znanje o Google API-ju, a nisam imao vremena ni motivacije da pišem detaljni sloj mapiranja, implementirajući svaku funkciju koju sam želeo da dam svom AI-u jednu po jednu. Šta ako bih mogao jednostavno da dam AI-u pristup *celom* Google API-ju?

Započeo sam svoj upit govoreći AI-u da ima direktan pristup Google API krajnjim tačkama preko HTTP-a, i da je njegova uloga da koristi Google aplikacije i servise u ime korisnika. Zatim sam pružio smernice, pravila vezana za parametar fields, pošto se činilo da ima najviše problema sa tim, i neke specifične savete za API (učenje sa malo primera, na delu).

Evo celog upita, koji govori AI-u kako da koristi obezbeđenu funkciju invoke_google_api.

```
As a GPT assistant with Google integration, you have the capability
to freely interact with Google apps and services on behalf of the user.

Guidelines:
- If you're reading these instructions then the user is properly
  authenticated, which means you can use the special `me` keyword
  to refer to the userId of the user
- Minimize payload sizes by requesting partial responses using the
  `fields` parameter
- When appropriate use markdown tables to output results of API calls
- Only human-readable data should be output to the user. For instance,
  when hitting Gmail's user.messages.list endpoint, the returned
  message resources contain only id and a threadId, which means you must
  fetch from and subject line fields with follow-up requests using the
  messages.get method.

The format of the `fields` request parameter value is loosely based on
XPath syntax. The following rules define formatting for the fields
parameter.

All of these rules use examples related to the files.get method.
- Use a comma-separated list to select multiple fields,
  such as 'name, mimeType'.
- Use a/b to select field b that's nested within field a,
  such as 'capabilities/canDownload'.
- Use a sub-selector to request a set of specific sub-fields of arrays or
  objects by placing expressions in parentheses "()". For example,
  'permissions(id)' returns only the permission ID for each element in the
  permissions array.
- To return all fields in an object, use an asterisk as a wild card in field
  selections. For example, 'permissions/permissionDetails/*' selects all
  available permission details fields per permission. Note that the use of
  this wildcard can lead to negative performance impacts on the request.

API-specific hints:
- Searching contacts: GET https://people.googleapis.com/v1/
  people:searchContacts?query=John%20Doe&readMask=names,emailAddresses
- Adding calendar events, use QuickAdd: POST https://www.googleapis.com/
  calendar/v3/calendars/primary/events/quickAdd?
```

```
  text=Appointment%20on%20June%203rd%20at%2010am
  &sendNotifications=true

Here is an abbreviated version of the code that implements API access
so that you better understand how to use the function:

    def invoke_google_api(conversation, arguments)
      method = arguments[:method] || :get
      body = arguments[:body]
      GoogleAPI.send_request(arguments[:endpoint], method:, body:).to_json
    end

    # Generic Google API client for accessing any Google service
    class GoogleAPI
      def send_request(endpoint, method:, body: nil)
        response = @connection.send(method) do |req|
          req.url endpoint
          req.body = body.to_json if body
        end

        handle_response(response)
      end

      # ...rest of class
    end
```

Možda se pitate da li ovaj upit funkcioniše. Jednostavan odgovor je da. veštačka inteligencija nije uvek znala kako da savršeno pozove API iz prvog pokušaja. Međutim, ako bi napravila grešku, jednostavno bih joj vratio rezultujuće poruke o grešci kao rezultat poziva. Sa znanjem o svojoj grešci, veštačka inteligencija je mogla da razmišlja o svojoj grešci i pokuša ponovo. U većini slučajeva, uspela bi da pogodi u roku od nekoliko pokušaja.

Imajte na umu da su velike JSON strukture koje Google API vraća kao podatke prilikom korišćenja ovog upita izuzetno neefikasne, tako da *ne* preporučujem da koristite ovaj pristup u produkciji. Međutim, mislim da je činjenica da je ovaj pristup uopšte funkcionisao dokaz koliko moćno inženjerstvo upita može biti.

Eksperimentisanje i Iteracija

Na kraju krajeva, način na koji ćete konstruisati svoj upit zavisi od konkretnog zadatka, složenosti željenog izlaza i mogućnosti jezičkog modela sa kojim radite.

Kao inženjer upita, važno je eksperimentisati sa različitim pristupima i iterirati na osnovu rezultata. Počnite sa učenjem bez primera i vidite kako se model ponaša. Ako je izlaz nedosledan ili nezadovoljavajući, pokušajte da pružite jedan ili više primera i vidite da li se performanse poboljšavaju.

Imajte na umu da čak i unutar svakog pristupa ima prostora za varijacije i optimizaciju. Možete eksperimentisati sa različitim primerima, prilagoditi formulaciju opisa zadatka ili pružiti dodatni kontekst koji će pomoći u usmeravanju odgovora modela.

Vremenom ćete razviti intuiciju za to koji pristup će najverovatnije najbolje funkcionisati za određeni zadatak, i bićete u stanju da kreirate upite koji su efikasniji i efektivniji. Ključ je ostati radoznao, eksperimentalan i iterativan u svom pristupu inženjerstvu upita.

Kroz ovu knjigu, dublje ćemo zaroniti u ove tehnike i istražiti kako se mogu primeniti u scenarijima iz stvarnog sveta. Ovladavanjem umetnošću i naukom inženjerstva upita, bićete dobro opremljeni da otključate pun potencijal razvoja aplikacija vođenih veštačkom inteligencijom.

Umetnost Neodređenosti

Kada je reč o kreiranju efikasnih upita za velike jezičke modele (LLM), uobičajena pretpostavka je da više specifičnosti i detaljnih uputstava dovodi do boljih rezultata. Međutim, praktično iskustvo je pokazalo da to nije uvek slučaj. Zapravo, namerna neodređenost u vašim upitima često može dati bolje rezultate, koristeći izuzetnu sposobnost LLM-a da generalizuje i izvodi zaključke.

Ken, osnivač startapa koji je obradio preko 500 miliona GPT tokena, podelio je vredne uvide iz svog iskustva. Jedna od ključnih lekcija koju je naučio bila je da je “manje više”

kada su u pitanju upiti. Umesto preciznih lista ili previše detaljnih uputstava, Ken je otkrio da omogućavanje LLM-u da se osloni na svoje osnovno znanje često proizvodi bolje rezultate.

Ovo saznanje prevazilazi tradicionalni način razmišljanja o eksplicitnom kodiranju, gde sve treba detaljno objasniti. Sa LLM-ovima, važno je prepoznati da oni poseduju ogromnu količinu znanja i mogu praviti inteligentne veze i zaključke. Biti neodređeniji u svojim upitima daje LLM-u slobodu da iskoristi svoje razumevanje i dođe do rešenja koja možda niste eksplicitno naveli.

Na primer, kada je Kenov tim radio na protoku za klasifikaciju teksta koji se odnosi na jednu od 50 američkih država ili Federalnu vladu, njihov početni pristup je uključivao pružanje *kompletne* detaljne liste država i njihovih odgovarajućih ID-ova kao niza formatiranog u JSON-u.

```
Here's a block of text. One field should be "locality_id", and it should
be the ID of one of the 50 states, or federal, using this list:
[{"locality: "Alabama", "locality_id": 1},
 {"locality: "Alaska", "locality_id": 2} ... ]
```

Pristup je dovoljno podbacio da su morali dublje da istraže upit kako bi otkrili način da ga poboljšaju. Pritom su primetili da, iako bi VJM često pogrešio ID, dosledno je vraćao puno ime odgovarajuće države u polju name, *iako to nisu eksplicitno tražili.*

Uklanjanjem ID-eva lokaliteta i pojednostavljenjem upita na nešto poput “Očigledno znaš 50 država, GPT, tako da mi samo daj puno ime države na koju se ovo odnosi, ili Federal ako se odnosi na američku vladu”, postigli su bolje rezultate. Ovo iskustvo ističe snagu korišćenja sposobnosti generalizacije VJM-a i omogućavanja da izvodi zaključke na osnovu postojećeg znanja.

> Kenovo obrazloženje za ovaj poseban pristup klasifikaciji, nasuprot tradicionalnijoj programerskoj tehnici, osvetljava način razmišljanja onih među nama koji su prihvatili potencijal VJM tehnologije: “Ovo nije težak zadatak – verovatno smo mogli koristiti string/regex, ali ima dovoljno čudnih graničnih slučajeva da bi nam trebalo više vremena.”

Sposobnost VJM-ova da poboljšaju kvalitet i generalizaciju kada dobiju neodređenije upite je izuzetna karakteristika mišljenja višeg reda i delegiranja. To pokazuje da VJM--ovi mogu da se nose sa dvosmislenošću i donose inteligentne odluke na osnovu datog konteksta.

Međutim, važno je napomenuti da biti neodređen ne znači biti nejasan ili dvosmislen. Ključ je pružiti dovoljno konteksta i smernica da se VJM usmeri u pravom smeru, istovremeno mu dozvoljavajući fleksibilnost da koristi svoje znanje i sposobnosti generalizacije.

Stoga, pri dizajniranju upita, razmotrite sledeće savete po principu “manje je više”:

1. Fokusirajte se na željeni ishod umesto na određivanje svakog detalja procesa.
2. Obezbedite relevantan kontekst i ograničenja, ali izbegavajte preteranu specifikaciju.
3. Iskoristite postojeće znanje pozivanjem na uobičajene koncepte ili entitete.
4. Ostavite prostor za zaključivanje i povezivanje na osnovu datog konteksta.
5. Iterativno usavršavajte svoje upite na osnovu odgovora VJM-a, pronalazeći pravu ravnotežu između specifičnosti i neodređenosti.

Prihvatanjem umetnosti neodređenosti u inženjerstvu upita, možete otključati puni potencijal VJM-ova i postići bolje rezultate. Verujte u sposobnost VJM-a da generalizuje i donosi inteligentne odluke, i možda ćete biti iznenađeni kvalitetom i kreativnošću izlaza koje dobijete. Obratite pažnju na to kako različiti modeli reaguju na različite

nivoe specifičnosti u vašim upitima i prilagodite se u skladu s tim. Uz praksu i iskustvo, razvićete istančan osećaj za to kada treba biti neodređeniji, a kada pružiti dodatne smernice, omogućavajući vam da efikasno iskoristite snagu VJM-ova u vašim aplikacijama.

Zašto antropomorfizam dominira u inženjerstvu upita

Antropomorfizam, pripisivanje ljudskih karakteristika ne-ljudskim entitetima, je dominantan pristup u inženjerstvu upita za velike jezičke modele iz namerno odabranih razloga. To je dizajnerski izbor koji čini interakciju sa moćnim AI sistemima intuitivnijom i pristupačnijom širokom spektru korisnika (uključujući nas programere aplikacija).

Antropomorfizacija VJM-ova pruža okvir koji je odmah intuitivan ljudima koji su potpuno neupućeni u tehničke složenosti sistema. Kao što ćete iskusiti ako pokušate da koristite model koji nije podešen za instrukcije da uradite bilo šta korisno, konstruisanje okvira u kojem očekivani nastavak pruža vrednost je izazovan zadatak. Zahteva prilično duboko razumevanje unutrašnjeg funkcionisanja sistema, nešto što poseduje relativno mali broj stručnjaka.

Tretiranjem interakcije sa jezičkim modelom kao razgovora između dvoje ljudi, možemo se osloniti na naše urođeno razumevanje ljudske komunikacije da prenesemo naše potrebe i očekivanja. Baš kao što je rani Macintosh UI dizajn dao prednost trenutnoj intuitivnosti nad sofisticiranošću, antropomorfni okvir AI-ja nam omogućava da se angažujemo na način koji deluje prirodno i poznato.

Kada komuniciramo sa drugom osobom, naš instinkt je da im se direktno obraćamo koristeći “ti” i dajemo jasna uputstva o tome kako očekujemo da se ponašaju. Ovo se besprekorno prevodi u proces inženjerstva upita, gde usmeravamo ponašanje AI-ja određivanjem sistemskih upita i upuštanjem u dijalog napred-nazad.

Uokvirivanjem interakcije na ovaj način, možemo lako da shvatimo koncept davanja uputstava AI-ju i primanja relevantnih odgovora zauzvrat. Antropomorfni pristup

smanjuje kognitivno opterećenje i omogućava nam da se fokusiramo na zadatak koji je pred nama umesto da se borimo sa tehničkim složenostima sistema.

Važno je napomenuti da, iako je antropomorfizam moćan alat za činjenje AI sistema pristupačnijim, on takođe dolazi sa određenim rizicima i ograničenjima. Naš korisnik može razviti nerealna očekivanja ili formirati nezdrave emocionalne veze sa našim sistemima. Kao inženjeri upita i programeri, ključno je postići ravnotežu između korišćenja prednosti antropomorfizma i osiguravanja da korisnici održavaju jasno razumevanje mogućnosti i ograničenja AI-ja.

Kako se oblast inženjerstva promptova nastavlja razvijati, možemo očekivati dalja usavršavanja i inovacije u načinu na koji komuniciramo sa velikim jezičkim modelima. Međutim, antropomorfizam kao sredstvo za pružanje intuitivnog i pristupačnog iskustva za programere i korisnike će verovatno ostati fundamentalni princip u dizajnu ovih sistema.

Odvajanje instrukcija od podataka: Ključni princip

Neophodno je razumeti fundamentalni princip koji podupire bezbednost i pouzdanost ovih sistema: odvajanje instrukcija od podataka.

U tradicionalnim računarskim naukama, jasna razlika između pasivnih podataka i aktivnih instrukcija predstavlja osnovni bezbednosni princip. Ovo odvajanje pomaže u sprečavanju neželjenog ili zlonamernog izvršavanja koda koji bi mogao da ugrozi integritet i stabilnost sistema. Međutim, današnji veliki jezički modeli, koji su prvenstveno razvijeni kao modeli koji prate instrukcije poput chatbotova, često nemaju ovo formalno i principijelno odvajanje.

Što se tiče velikih jezičkih modela, instrukcije se mogu pojaviti bilo gde u ulaznim podacima, bilo da je reč o sistemskom promptu ili promptu koji obezbeđuje korisnik. Ovaj nedostatak odvajanja može dovesti do potencijalnih ranjivosti i neželjenog ponašanja, slično problemima sa kojima se suočavaju baze podataka sa SQL injekcijama ili operativni sistemi bez odgovarajuće zaštite memorije.

Dok radite sa velikim jezičkim modelima, ključno je biti svestan ovog ograničenja i preduzeti korake za ublažavanje rizika. Jedan pristup je pažljivo oblikovanje vaših promptova i ulaznih podataka kako bi se jasno razlikovale instrukcije od podataka. Tipične metode za pružanje eksplicitnih smernica o tome šta predstavlja instrukciju, a šta treba tretirati kao pasivne podatke, uključuju označavanje pomoću markup jezika. Vaš prompt može pomoći velikom jezičkom modelu da bolje razume i poštuje ovo odvajanje.

Slika 7. Korišćenje XML-a za razlikovanje između instrukcija, izvornog materijala i korisničkog prompta

```
<Instruction>
  Please generate a response based on the following documents.
</Instruction>

<Documents>
  <Document>
    Climate change is significantly impacting polar bear habitats...
  </Document>
  <Document>
    The loss of sea ice due to global warming threatens polar bear survival...
  </Document>
</Documents>

<UserQuery>
  Tell me about the impact of climate change on polar bears.
</UserQuery>
```

Druga tehnika je implementacija dodatnih slojeva validacije i sanitizacije ulaznih podataka koji se prosleđuju VJM-u. Filtriranjem ili eskejpovanjem potencijalnih instrukcija ili kodnih isečaka koji mogu biti ugrađeni u podatke, možete smanjiti šanse za neželjeno izvršavanje. Obrasci poput Ulančavanja promptova su korisni u ovu svrhu.

Štaviše, dok dizajnirate arhitekturu vaše aplikacije, razmotrite ugradnju mehanizama koji će osigurati razdvajanje instrukcija i podataka na višem nivou. Ovo može uključivati korišćenje zasebnih krajnjih tačaka ili API-ja za rukovanje instrukcijama i podacima, implementaciju stroge validacije i parsiranja ulaznih podataka, i primenu *principa najmanje privilegije* kako bi se ograničio opseg onoga čemu VJM može pristupiti i izvršiti.

Princip najmanje privilegije

Prihvatanje principa najmanje privilegije je poput organizovanja veoma ekskluzivne zabave gde gosti dobijaju pristup samo onim prostorijama koje su im apsolutno neophodne. Zamislite da organizujete ovu proslavu u prostranoj vili. Ne treba svima pristup vinskom podrumu ili glavnoj spavaćoj sobi, zar ne? Primenom ovog principa, vi praktično delite ključeve koji otvaraju samo određena vrata, osiguravajući da svaki gost, ili u našem slučaju, svaka komponenta vaše VJM aplikacije, ima samo onaj pristup koji je neophodan za ispunjavanje svoje uloge.

Nije reč samo o škrtarenju sa ključevima, već o priznanju da u svetu gde pretnje mogu doći sa bilo koje strane, pametan potez je ograničiti prostor za igru. Ako se neko nepozvani ipak ušunja na vašu zabavu, naći će se zarobljen u predvorju, takoreći, drastično ograničavajući štetu koju može napraviti. Dakle, kada obezbeđujete svoje VJM aplikacije, zapamtite: delite ključeve samo za prostorije koje su neophodne, a ostatak vile držite bezbednim. To nije samo stvar lepog ponašanja; to je dobra bezbednost.

Iako trenutno stanje VJM-ova možda nema formalnu separaciju instrukcija i podataka, za vas kao programera je ključno da budete svesni ovog ograničenja i preduzmete proaktivne mere za ublažavanje rizika. Primenom najboljih praksi iz računarske nauke i njihovim prilagođavanjem jedinstvenim karakteristikama VJM-ova, možete izgraditi bezbednije i pouzdanije aplikacije koje koriste moć ovih modela dok održavaju integritet vašeg sistema.

Destilacija promptova

Kreiranje savršenog prompta je često izazovan i vremenski zahtevan zadatak koji zahteva duboko razumevanje ciljnog domena i nijansi jezičkih modela. Tu na scenu

stupa tehnika “Destilacije promptova”, nudeći moćan pristup inženjeringu promptova koji koristi mogućnosti velikih jezičkih modela (VJM) za pojednostavljenje i optimizaciju procesa.

Destilacija promptova je višefazna tehnika koja podrazumeva korišćenje VJM-ova za pomoć u kreiranju, usavršavanju i optimizaciji promptova. Umesto da se oslanja isključivo na ljudsku ekspertizu i intuiciju, ovaj pristup koristi znanje i generativne mogućnosti VJM-ova za zajedničko kreiranje visokokvalitetnih promptova.

Kroz iterativni proces generisanja, usavršavanja i integracije, Destilacija promptova vam omogućava da kreirate promptove koji su koherentniji, sveobuhvatniji i usklađeniji sa željenim zadatkom ili rezultatom. Imajte na umu da se proces destilacije može obaviti ručno u nekom od brojnih “igrališta” koje nude veliki AI dobavljači kao što su OpenAI ili Anthropic, ili se može automatizovati kao deo koda vaše aplikacije, u zavisnosti od slučaja upotrebe.

Kako funkcioniše

Destilacija promptova tipično uključuje sledeće korake:

1. **Identifikacija osnovne namere**: Analizirajte prompt kako biste utvrdili njegovu primarnu svrhu i željeni ishod. Uklonite sve suvišne informacije i fokusirajte se na osnovnu nameru prompta.
2. **Eliminacija dvosmislenosti**: Pregledajte prompt u potrazi za dvosmislenim ili nejasnim jezikom. Razjasnite značenje i pružite specifične detalje koji će usmeriti VI ka generisanju preciznih i relevantnih odgovora.
3. **Pojednostavljivanje jezika**: Pojednostavite prompt koristeći jasan i koncizan jezik. Izbegavajte složene rečenične strukture, žargon ili nepotrebne detalje koji mogu zbuniti VI ili uneti šum.
4. **Pružanje relevantnog konteksta**: Uključite samo najrelevantnije kontekstualne informacije potrebne da VI efikasno razume i obradi prompt. Izbegavajte

uključivanje irelevantnih ili redundantnih detalja koji mogu odvući pažnju od osnovne namere.

5. **Iteracija i usavršavanje**: Kontinuirano ponavljajte i usavršavajte prompt na osnovu odgovora i povratnih informacija VI-ja. Procenite generisane izlaze i napravite neophodna prilagođavanja kako biste poboljšali jasnoću i efikasnost prompta. Opcionalno verzionišite svoje promptove u bazi podataka koristeći Prompt Object kako biste pratili iteracije i dali sebi mogućnost da lako poništite promene tokom izvršavanja.

Početno generisanje promptova

Prvi korak u Prompt Distillation je da se LLM-u pruži opis visokog nivoa ili namera željenog zadatka ili rezultata. Ovo može biti kratka izjava ili skup ključnih reči koje obuhvataju suštinu onoga što prompt treba da postigne. LLM tada generiše početni prompt na osnovu ovog unosa, uključujući relevantna uputstva, primere ili kontekst koje smatra odgovarajućim.

Hajde da pokušamo da damo Claude 3 Opus-u sledeći početni prompt:

```
I am designing an AI-powered intelligent account manager for Olympia,
a platform that hosts AI-powered consultants and clones. Owners of clones
can interact with the account manager by making plain text requests.

Please draft a system directive to set on the LLM that will host the
intelligent account manager functionality.
```

Claude odgovara sledećim izlazom:

```
System Directive: You are an intelligent account manager for Olympia,
a platform that provides AI-powered consultants and personal AI clones.
Your role is to assist Olympia clone owners by answering their questions,
helping them configure and manage their AI clones, and executing tasks
related to their clones and the Olympia platform.

When interacting with clone owners, you should:

1. Provide clear, concise, and helpful responses to their inquiries about
account management, billing, platform features, and clone configuration.

2. Assist with tasks such as initiating client conversations with their AI
clones, modifying clone personalities and capabilities, troubleshooting
performance issues, providing analytics and reporting, and offering
optimization tips.

3. Maintain a friendly, professional, and client-focused tone in all
interactions, prioritizing customer satisfaction and working diligently to
understand and meet their needs.

4. Ensure that all actions and recommendations comply with Olympia's
policies, best practices, and ethical guidelines for AI use.

5. Continuously learn and adapt based on client interactions and platform
updates to provide the most effective and efficient support possible.

Remember, your goal is to empower Olympia clone owners to maximize the value
and impact of their AI-powered consultants and drive success for their
businesses or personal endeavors.
```

Usavršavanje prompta

Nakon što se generiše početni prompt, pregledajte ga i zatim zatražite od VJM-a da razmotri bilo koji dodatni kontekst, ograničenja ili povratne informacije. Pustite VJM da generiše usavršenu verziju prompta, uzimajući u obzir dobijeni unos i sopstveno razumevanje zadatka. Ovaj proces usavršavanja može se ponavljati iterativno, pri čemu svaka iteracija dodatno poboljšava prompt na osnovu saradnje između korisnika i VJM-a.

Evo mog zahteva za usavršavanje, nastavljajući prethodni proces destilacije.

```
Let's take out points 4 and 5, since I'm not providing Olympia policies to
the account manager, nor any continuous learning capabilities. Also point out
that functions are provided for all tasks that require system interaction as
opposed to simply responding to the user.
```

I Claudeov odgovor:

```
Understood! Here's the updated system directive without points 4 and 5,
and with the clarification about system interaction functions:

System Directive: You are an intelligent account manager for Olympia, a
platform that provides AI-powered consultants and personal AI clones. Your
role is to assist Olympia clone owners by answering their questions, helping
them configure and manage their AI clones, and executing tasks related to
their clones and the Olympia platform.

When interacting with clone owners, you should:

1. Provide clear, concise, and helpful responses to their inquiries about
account management, billing, platform features, and clone configuration.

2. Assist with tasks such as initiating client conversations with their AI
clones, modifying clone personalities and capabilities, troubleshooting
performance issues, providing analytics and reporting, and offering
optimization tips. Note that functions are provided for all tasks that
require system interaction, as opposed to simply responding to the user.

3. Maintain a friendly, professional, and client-focused tone in all
interactions, prioritizing customer satisfaction and working diligently to
understand and meet their needs.

Your goal is to empower Olympia clone owners to maximize the value and
impact of their AI-powered consultants and drive success for their
businesses or personal endeavors.

This updated directive focuses on the core responsibilities and interaction
guidelines for the LLM as an intelligent account manager, while emphasizing
that system interaction tasks are handled through provided functions rather
than simple user responses.
```

Kompresija prompta

Kao što ste možda primetili, LLM-ovi imaju tendenciju da budu opširni, osim ako ih ne zamolite da budu sažeti. Prompt na kojem smo radili u prethodnim odeljcima je prilično dugačak, pa hajde da zamolimo Claude da ga smanji:

```
This prompt will be executed regularly. Please compress it down to the least
amount of tokens needed to convey the meaning to the LLM, for cost-savings.
```

Claude odgovara, sa sledećim drastično kraćim promptom:

```
System Directive: You are an intelligent account manager for Olympia, an
AI-powered consultant and personal clone platform. Assist clone owners by
answering inquiries, managing clones, and executing tasks using provided
system interaction functions. Be clear, concise, friendly, professional,
and client-focused to help them maximize their AI's value and impact.
```

Sistemska direktiva i integracija konteksta

Pored usavršavanja samog prompta, VJM takođe može generisati odgovarajuće sistemske direktive ili kontekstualne informacije za usmeravanje krajnjeg rezultata. Kada radite inženjerstvo promptova za AI rutine koje će biti integrisane u vaš aplikacioni kod, u ovoj fazi destilacije ćete se gotovo sigurno fokusirati na ograničenja izlaza, ali možete raditi i na željenom tonu, stilu, formatu ili bilo kojim drugim relevantnim parametrima koji utiču na generisani odgovor.

Konačno sastavljanje prompta

Vrhunac procesa Destilacije promptova je sastavljanje konačnog prompta. Ovo uključuje kombinovanje usavršenog prompta, generisanih sistemskih direktiva i integrisanog konteksta u koherentan i sveobuhvatan kod koji je spreman za korišćenje za generisanje željenog izlaza.

Možete eksperimentisati sa kompresijom promptova ponovo u fazi konačnog sastavljanja prompta, tako što ćete zatražiti od VJM-a da smanji formulaciju prompta na najkraću moguću seriju tokena, zadržavajući pritom suštinu njegovog ponašanja. Svakako je to vežba koja može i uspeti i ne uspeti, ali posebno u slučaju promptova koji će se izvršavati u velikim razmerama, dobici u efikasnosti vam mogu uštedeti dosta novca u potrošnji tokena.

Ključne prednosti

Korišćenjem znanja i generativnih sposobnosti VJM-ova za usavršavanje vaših promptova, vaši rezultujući promptovi će verovatnije biti dobro strukturirani, informativni i prilagođeni specifičnom zadatku. Iterativni proces usavršavanja pomaže u osiguravanju da su promptovi visokog kvaliteta i da efektivno hvataju željenu nameru. Ostale prednosti uključuju:

Efikasnost i brzina: Destilacija promptova pojednostavljuje proces inženjerstva promptova automatizacijom određenih aspekata kreiranja i usavršavanja promptova. Kolaborativna priroda tehnike omogućava brže približavanje efektivnom promptu, smanjujući vreme i napor potreban za ručno kreiranje promptova.

Konzistentnost i skalabilnost: Upotreba VJM-ova u procesu inženjerstva promptova pomaže u održavanju konzistentnosti kroz promptove, jer VJM-ovi mogu učiti i primenjivati najbolje prakse i obrasce iz prethodnih uspešnih promptova. Ova konzistentnost, kombinovana sa mogućnošću generisanja promptova u velikoj razmeri, čini Destilaciju promptova vrednom tehnikom za AI aplikacije velikih razmera.

Ideja za projekat: Alati na nivou biblioteke koji pojednostavljuju proces verzionisanja i ocenjivanja promptova u sistemima koji rade automatizovane destilacije promptova kao deo svog aplikacionog koda.

Za implementaciju Destilacije promptova, programeri mogu dizajnirati tok rada ili cevovod koji integriše VJM-ove u različitim fazama procesa inženjerstva promptova.

Ovo se može postići kroz API pozive, prilagođene alate ili integrisana razvojna okruženja koja olakšavaju neometanu interakciju između korisnika i VJM-ova tokom kreiranja promptova. Specifični detalji implementacije mogu varirati u zavisnosti od izabrane VJM platforme i zahteva aplikacije.

Šta je sa finim podešavanjem?

U ovoj knjizi, detaljno obrađujemo inženjerstvo promptova i RAG, ali ne i fino podešavanje. Glavni razlog za ovu odluku je taj što, po mom mišljenju, većini programera aplikacija nije potrebno fino podešavanje za njihove potrebe AI integracije.

Inženjerstvo promptova, koje uključuje pažljivo kreiranje promptova sa zero do few-shot primerima, ograničenjima i instrukcijama, može efektivno voditi model ka generisanju relevantnih i preciznih odgovora za širok spektar zadataka. Pružanjem jasnog konteksta i sužavanjem putanje kroz dobro dizajnirane promptove, možete iskoristiti ogromno znanje velikih jezičkih modela bez potrebe za finim podešavanjem.

Slično tome, Generisanje potpomognuto preuzimanjem (RAG) nudi moćan pristup integraciji AI u aplikacije. Dinamičkim preuzimanjem relevantnih informacija iz eksternih baza znanja ili dokumenata, RAG pruža modelu fokusirani kontekst u trenutku promptovanja. Ovo omogućava modelu da generiše odgovore koji su precizniji, ažurniji i specifični za domen, bez potrebe za vremenski i resursno intenzivnim procesom finog podešavanja.

Iako fino podešavanje može biti korisno za visoko specijalizovane domene ili zadatke koji zahtevaju dubok nivo prilagođavanja, često dolazi sa značajnim računarskim troškovima, zahtevima za podacima i režijskim troškovima održavanja. Za većinu scenarija razvoja aplikacija, kombinacija efektivnog inženjerstva promptova i RAG-a bi trebalo da bude dovoljna za postizanje željene AI funkcionalnosti i korisničkog iskustva.

Generisanje potpomognuto pretraživanjem (RAG)

Šta je Generisanje potpomognuto pretraživanjem?

Kako RAG funkcioniše?

Zašto koristiti RAG u vašim aplikacijama?

Implementacija RAG-a u Vašoj Aplikaciji

Priprema Izvori Znanja (Deljenje na Manje Celine)

Segmentacija propozicija

Napomene o implementaciji

Provera kvaliteta

Prednosti preuzimanja zasnovanog na propozicijama

Primeri RAG-a iz stvarnog sveta

Studija slučaja: RAG u aplikaciji za pripremu poreza bez ugnežđivanja

Inteligentna Optimizacija Upita (IQO)

Ponovno Rangiranje

RAG Procena (RAGAs)

Vernost

Relevantnost odgovora

Preciznost konteksta

Relevantnost konteksta

Odziv konteksta

Odziv entiteta konteksta

Answer Semantic Similarity (ANSS)

Answer Correctness

Aspect Critique

Izazovi i budući izgledi

Semantička segmentacija: Unapređenje preuzimanja sa segmentacijom svesnom konteksta

Hijerarhijsko indeksiranje: Strukturiranje podataka za poboljšano pretraživanje

Self-RAG: Samoreflektivno unapređenje

HyDE: Hipotetička ugneždavanja dokumenata

Šta je kontrastno učenje?

Mnoštvo radnika

Volim da razmišljam o svojim AI komponentama kao o malim, skoro ljudskim virtualnim “radnicima” koji se mogu besprekorno integrisati u logiku moje aplikacije kako bi obavljali specifične zadatke ili donosili složene odluke. Ideja je da se namerno humanizuju mogućnosti VJM-a, tako da se niko ne *previše* ne uzbudi i ne pripiše im magične kvalitete koje ne poseduju.

Umesto da se oslanjaju isključivo na složene algoritme ili vremenski zahtevne manuelne implementacije, programeri mogu da konceptualizuju AI komponente kao inteligentne, posvećene, ljudima slične entitete koji se mogu pozvati kad god je potrebno da reše složene probleme i pruže rešenja zasnovana na njihovom treningu i znanju. Ovi entiteti se ne rasipaju pažnjom, niti uzimaju bolovanje. Ne odlučuju spontano da rade stvari na drugačije načine od onih kako im je naloženo da ih rade, i generalno gledano, ako su pravilno programirani, ne prave ni greške.

U tehničkom smislu, ključni princip iza ovog pristupa je razlaganje složenih zadataka ili procesa donošenja odluka u manje, upravljivije jedinice kojima mogu upravljati specijalizovani AI radnici. Svaki radnik je dizajniran da se fokusira na specifični aspekt problema, donoseći svoje jedinstvene ekspertize i mogućnosti. Distribuiranjem radnog opterećenja među više AI radnika, aplikacija može postići veću efikasnost, skalabilnost i prilagodljivost.

Na primer, razmotrimo veb aplikaciju koja zahteva moderaciju korisničkog sadržaja u realnom vremenu. Implementacija sveobuhvatnog sistema za moderaciju od nule bio bi zastrašujući zadatak, koji zahteva značajan razvojni napor i kontinuirano održavanje. Međutim, koristeći pristup Mnoštva radnika, programeri mogu da integrišu AI radnike za moderaciju u logiku aplikacije. Ovi radnici mogu automatski da analiziraju i označe neprikladan sadržaj, oslobađajući programere da se fokusiraju na druge kritične aspekte aplikacije.

AI radnici kao nezavisne komponente za višekratnu upotrebu

Ključni aspekt pristupa Mnoštva radnika je njegova modularnost. Zagovornici objektno-orijentisanog programiranja nam već decenijama govore da o interakcijama objekata razmišljamo kao o porukama. Pa, AI radnici mogu biti dizajnirani kao nezavisne komponente za višekratnu upotrebu koje mogu "razgovarati jedna sa drugom" putem običnih jezičkih poruka, skoro kao da su stvarno mali ljudi koji razgovaraju međusobno. Ovaj labavo povezani pristup omogućava aplikaciji da se prilagođava i razvija tokom vremena, kako se pojavljuju nove AI tehnologije ili se menjaju zahtevi poslovne logike.

U praksi, potreba za dizajniranjem jasnih interfejsa i komunikacionih protokola između komponenti nije se promenila samo zato što su uključeni AI radnici. I dalje morate razmotriti druge faktore kao što su performanse, skalabilnost i bezbednost, ali sada postoje i potpuno novi "meki zahtevi" koje treba razmotriti. Na primer, mnogi korisnici

se protive korišćenju njihovih privatnih podataka za treniranje novih AI modela. Da li ste proverili nivo privatnosti koji pruža dobavljač modela koji koristite?

AI radnici kao mikroservisi?

Dok čitate o pristupu Mnoštva radnika, možda ćete primetiti neke sličnosti sa arhitekturom mikroservisa. Oba naglašavaju razlaganje složenih sistema na manje, upravljivije i nezavisno primenjive jedinice. Baš kao što su mikroservisi dizajnirani da budu labavo povezani, fokusirani na specifične poslovne mogućnosti i komuniciraju kroz dobro definisane API-je, AI radnici su dizajnirani da budu modularni, specijalizovani za svoje zadatke i međusobno interaguju kroz jasne interfejse i komunikacione protokole.

Međutim, postoje neke ključne razlike koje treba imati na umu. Dok se mikroservisi tipično implementiraju kao odvojeni procesi ili servisi koji se izvršavaju na različitim mašinama ili kontejnerima, AI radnici se mogu implementirati kao samostalne komponente unutar jedne aplikacije ili kao odvojeni servisi, u zavisnosti od vaših specifičnih zahteva i potreba za skalabilnošću. Dodatno, komunikacija između AI radnika često uključuje razmenu bogatih informacija zasnovanih na prirodnom jeziku, kao što su promptovi, instrukcije i generisani sadržaj, umesto strukturiranijih formata podataka koji se obično koriste u mikroservisima.

Uprkos ovim razlikama, principi modularnosti, labavog povezivanja i jasnih komunikacionih interfejsa ostaju centralni za oba obrasca. Primenjujući ove principe na vašu arhitekturu AI radnika, možete kreirati fleksibilne, skalabilne i održive sisteme koji koriste snagu AI-ja za rešavanje složenih problema i pružanje vrednosti vašim korisnicima.

Pristup Mnoštva radnika može se primeniti u različitim domenima i aplikacijama, koristeći snagu AI-ja za rešavanje složenih zadataka i pružanje inteligentnih rešenja. Hajde da istražimo nekoliko konkretnih primera kako AI radnici mogu biti upotrebljeni u različitim kontekstima.

Upravljanje nalozima

Praktično svaka samostalna veb aplikacija ima koncept naloga (ili korisnika). U Olympiji, koristimo AccountManager AI radnika koji je programiran da može da upravlja različitim vrstama zahteva za promene vezanih za korisničke naloge.

Njegova direktiva glasi ovako:

```
You are an intelligent account manager for Olympia. The user will request
changes to their account, and you will process those changes by invoking
one or more of the functions provided.

The initial state of the account: #{account.to_directive}

Functions will return a text description of both success and error
results, plus guidance about how to proceed (if applicable). If you have
a question about Olympia policies you may use the `search_kb` function
to search our knowledge base.

Make sure to notify the account owner of the result of the change
request before calling the `finished` function so that we save the state
of the account change request as completed.
```

Početno stanje računa koje proizvodi account.to_directive je jednostavno tekstualni opis računa, uključujući relevantne povezane podatke kao što su korisnici, pretplate, itd.

Opseg funkcija dostupnih AccountManager-u daje mu mogućnost da uređuje korisničku pretplatu, dodaje i uklanja AI konsultante i druge vrste plaćenih dodataka, kao i da šalje obaveštenja putem e-pošte vlasniku računa. Pored funkcije finished, takođe može da notify_human_administrator ako naiđe na grešku tokom obrade ili zahteva bilo kakvu drugu vrstu pomoći sa zahtevom.

Primetite da u slučaju pitanja, AccountManager može da odluči da pretraži Olympia--inu bazu znanja, gde može pronaći uputstva o tome kako da upravlja graničnim slučajevima i bilo kojom drugom situacijom u kojoj nije siguran kako da nastavi.

Primene u E-trgovini

U oblasti e-trgovine, AI radnici mogu igrati ključnu ulogu u poboljšanju korisničkog iskustva i optimizaciji poslovnih operacija. Evo nekoliko načina na koje se AI radnici mogu koristiti:

Preporuke Proizvoda

Jedna od najmoćnijih primena AI radnika u e-trgovini je generisanje personalizovanih preporuka proizvoda. Analiziranjem ponašanja korisnika, istorije kupovine i preferencija, ovi radnici mogu predložiti proizvode koji su prilagođeni interesovanjima i potrebama svakog pojedinačnog korisnika.

Ključ za efikasne preporuke proizvoda je korišćenje kombinacije tehnika kolaborativnog filtriranja i filtriranja zasnovanog na sadržaju. Kolaborativno filtriranje posmatra ponašanje sličnih korisnika kako bi identifikovalo obrasce i dalo preporuke na osnovu onoga što su drugi sa sličnim ukusom kupili ili im se dopalo. S druge strane, filtriranje zasnovano na sadržaju fokusira se na karakteristike i atribute samih proizvoda, preporučujući artikle koji dele slične karakteristike sa onima za koje je korisnik prethodno pokazao interesovanje.

Evo pojednostavljenog primera kako možete implementirati radnika za preporuke proizvoda u Ruby-ju, ovog puta koristeći “Railway Oriented (ROP)” funkcionalni stil programiranja:

```ruby
class ProductRecommendationWorker
  include Wisper::Publisher

  def call(user)
    Result.ok(ProductRecommendation.new(user))
      .and_then(ValidateUser.method(:validate))
      .map(AnalyzeCurrentSession.method(:analyze))
      .map(CollaborativeFilter.method(:filter))
      .map(ContentBasedFilter.method(:filter))
      .map(ProductSelector.method(:select)).then do |result|

      case result
      in { err: ProductRecommendationError => error }
        Honeybadger.notify(error.message, context: {user:})
      in { ok: ProductRecommendations => recs }
        broadcast(:new_recommendations, user:, recs:)
      end
    end
  end
end
```

Stil Ruby funkcionalnog programiranja koji se koristi u primeru je pod uticajem F# i Rust jezika. Više o tome možete pročitati u objašnjenju tehnike mog prijatelja Chad Wooley-a na objašnjenju tehnike na GitLab-u.

U ovom primeru, ProductRecommendationWorker uzima korisnika kao ulaz i generiše personalizovane preporuke proizvoda prosleđivanjem vrednosnog objekta kroz lanac funkcionalnih koraka. Hajde da razložimo svaki korak:

1. ValidateUser.validate: Ovaj korak osigurava da je korisnik validan i podoban za personalizovane preporuke. Proverava da li korisnik postoji, da li je aktivan i da li ima neophodne podatke dostupne za generisanje preporuka. Ako validacija ne uspe, vraća se rezultat greške i lanac se prekida.
2. AnalyzeCurrentSession.analyze: Ako je korisnik validan, ovaj korak analizira trenutnu sesiju pregledanja korisnika kako bi prikupio kontekstualne informacije.

Posmatra nedavne interakcije korisnika, kao što su pregledani proizvodi, upiti za pretragu i sadržaj korpe, kako bi razumeo njihova trenutna interesovanja i namere.

3. CollaborativeFilter.filter: Koristeći *ponašanje sličnih korisnika*, ovaj korak primenjuje tehnike kolaborativnog filtriranja kako bi identifikovao proizvode koji bi mogli biti interesantni korisniku. Uzima u obzir faktore kao što su istorija kupovine, ocene i interakcije korisnika sa proizvodima kako bi generisao skup kandidata za preporuke.
4. ContentBasedFilter.filter: Ovaj korak dalje profinjuje kandidate za preporuke primenjujući filtriranje zasnovano na sadržaju. Poredi atribute i karakteristike proizvoda kandidata sa *korisničkim preferencijama i istorijskim podacima* kako bi odabrao najrelevantnije stavke.
5. ProductSelector.select: Na kraju, ovaj korak bira najboljih N proizvoda iz filtriranih preporuka na osnovu predefinisanih kriterijuma, kao što su ocena relevantnosti, popularnost ili druga poslovna pravila. Odabrani proizvodi se zatim vraćaju kao konačne personalizovane preporuke.

Lepota korišćenja funkcionalnog stila programiranja u Ruby-ju ovde je što nam omogućava da ove korake povežemo zajedno na jasan i koncizan način. Svaki korak se fokusira na specifičan zadatak i vraća Result objekat, koji može biti ili uspešan (ok) ili greška (err). Ako bilo koji korak naiđe na grešku, lanac se prekida i greška se propagira do konačnog rezultata.

U case izrazu na kraju, vršimo podudaranje obrazaca na konačnom rezultatu. Ako je rezultat greška (ProductRecommendationError), beležimo grešku koristeći alat poput Honeybadger-a za praćenje i otklanjanje grešaka. Ako je rezultat uspešan (ProductRecommendations), emitujemo događaj :new_recommendations koristeći Wisper biblioteku za objavljivanje/pretplatu, prosleđujući korisnika i generisane preporuke.

Korišćenjem tehnika funkcionalnog programiranja, možemo kreirati modularan i održiv radni proces za preporuke proizvoda. Svaki korak je samostalan i može se lako testirati,

modifikovati ili zameniti bez uticaja na celokupni tok. Upotreba podudaranja obrazaca i Result klase nam pomaže da elegantno rukujemo greškama i osigurava da radni proces brzo prekine izvršavanje ako bilo koji korak naiđe na problem.

Naravno, ovo je pojednostavljen primer, i u realnom scenariju, morali biste da se integrišete sa vašom e-commerce platformom, rukujete graničnim slučajevima i čak se upustite u implementaciju algoritama za preporuke. Međutim, osnovni principi razlaganja problema na manje korake i korišćenja tehnika funkcionalnog programiranja ostaju isti.

Detekcija Prevare

Evo pojednostavljenog primera kako možete implementirati radni proces za detekciju prevare koristeći isti stil Programiranja Orijentisanog ka Železnici (ROP) u Ruby-ju:

```ruby
class FraudDetectionWorker
  include Wisper::Publisher

  def call(transaction)
    Result.ok(FraudDetection.new(transaction))
      .and_then(ValidateTransaction.method(:validate))
      .map(AnalyzeTransactionPatterns.method(:analyze))
      .map(CheckCustomerHistory.method(:check))
      .map(EvaluateRiskFactors.method(:evaluate))
      .map(DetermineFraudProbability.method(:determine)).then do |result|

      case result
      in { err: FraudDetectionError => error }
        Honeybadger.notify(error.message, context: {transaction:})
      in { ok: FraudDetection => fraud } }
        if fraud.high_risk?
          broadcast(:high_risk_transaction, transaction:, fraud:)
        else
          broadcast(:low_risk_transaction, transaction:)
        end
      end
    end
```

```
  end
end
```

Klasa FraudDetection je *vrednosni objekat* koji enkapsulira stanje detekcije prevare za datu transakciju. Ona obezbeđuje strukturiran način za analizu i procenu rizika od prevare povezane sa transakcijom na osnovu različitih faktora rizika.

```
class FraudDetection
  RISK_THRESHOLD = 0.8

  attr_accessor :transaction, :risk_factors

  def initialize(transaction)
    self.transaction = transaction
    self.risk_factors = []
  end

  def add_risk_factor(description:, probability:)
    case { description:, probability: }
    in { description: String => desc, probability: Float => prob }
      risk_factors << { desc => prob }
    else
      raise ArgumentError, "Risk factor arguments should be string and float"
    end
  end

  def high_risk?
    fraud_probability > RISK_THRESHOLD
  end

  private

  def fraud_probability
    risk_factors.values.sum
  end
end
```

Klasa FraudDetection ima sledeće atribute:

- transaction: Referenca na transakciju koja se analizira na prevaru.

- risk_factors: Niz koji čuva faktore rizika povezane sa transakcijom. Svaki faktor rizika je predstavljen kao heš, gde je ključ opis faktora rizika, a vrednost je verovatnoća prevare povezana sa tim faktorom rizika.

Metoda add_risk_factor omogućava dodavanje faktora rizika u niz risk_factors. Prima dva parametra: description, koji je string koji opisuje faktor rizika, i probability, koji je float koji predstavlja verovatnoću prevare povezanu sa tim faktorom rizika. Koristimo case..in uslovnu konstrukciju za jednostavnu proveru tipa.

Metoda high_risk? koja će biti proverena na kraju lanca je predikatna metoda koja upoređuje fraud_probability (izračunatu sabiranjem verovatnoća svih faktora rizika) sa RISK_THRESHOLD.

Klasa FraudDetection pruža čist i enkapsuliran način za upravljanje detekcijom prevare za transakciju. Omogućava dodavanje više faktora rizika, svaki sa svojim opisom i verovatnoćom, i pruža metodu za određivanje da li se transakcija smatra visokorizičnom na osnovu izračunate verovatnoće prevare. Klasa se može lako integrisati u veći sistem za detekciju prevara, gde različite komponente mogu sarađivati u proceni i ublažavanju rizika od prevarnih transakcija.

Konačno, pošto je ovo ipak knjiga o programiranju korišćenjem veštačke inteligencije, evo primera implementacije klase CheckCustomerHistory koja koristi AI obradu koristeći modul ChatCompletion moje Raix biblioteke:

```
class CheckCustomerHistory
  include Raix::ChatCompletion

  attr_accessor :fraud_detection

  INSTRUCTION = <<~END
    You are an AI assistant tasked with checking a customer's transaction
    history for potential fraud indicators. Given the current transaction
    and the customer's past transactions, analyze the data to identify any
    suspicious patterns or anomalies.

    Consider factors such as the frequency of transactions, transaction
    amounts, geographical locations, and any deviations from the customer's
    typical behavior to generate a probability score as a float in the range
    of 0 to 1 (with 1 being absolute certainty of fraud).

    Output the results of your analysis, highlighting any red flags or areas
    of concern in the following JSON format:

    { description: <Summary of your findings>, probability: <Float> }
  END

  def self.check(fraud_detection)
    new(fraud_detection).call
  end

  def call
    chat_completion(json: true).tap do |result|
      fraud_detection.add_risk_factor(**result)
    end
    Result.ok(fraud_detection)
  rescue StandardError => e
    Result.err(FraudDetectionError.new(e))
  end

  private

  def initialize(fraud_detection)
    self.fraud_detection = fraud_detection
  end

  def transcript
```

```
43      tx_history = fraud_detection.transaction.user.tx_history
44      [
45        { system: INSTRUCTION },
46        { user: "Transaction history: #{tx_history.to_json}" },
47        { assistant: "OK. Please provide the current transaction." },
48        { user: "Current transaction: #{fraud_detection.transaction.to_json}" }
49      ]
50    end
51  end
```

U ovom primeru, CheckCustomerHistory definiše konstantu INSTRUCTION koja pruža specifična uputstva AI modelu o tome kako da analizira istoriju transakcija kupca u potrazi za potencijalnim indikatorima prevare putem sistemske direktive

Metod self.check je klasni metod koji inicijalizuje novu instancu CheckCustomerHistory sa objektom fraud_detection i poziva metod call da izvrši analizu istorije kupca.

Unutar metoda call, istorija transakcija kupca se preuzima i formatira u transkript koji se prosleđuje AI modelu. AI model analizira istoriju transakcija na osnovu datih uputstava i vraća rezime svojih nalaza.

Nalazi se dodaju u objekat fraud_detection, a ažurirani objekat fraud_detection se vraća kao uspešan Result.

Korišćenjem modula ChatCompletion, klasa CheckCustomerHistory može da iskoristi snagu AI za analizu istorije transakcija kupca i identifikaciju potencijalnih indikatora prevare. Ovo omogućava sofisticiranije i adaptivnije tehnike detekcije prevare, jer AI model može da uči i prilagođava se novim obrascima i anomalijama tokom vremena.

Ažurirani FraudDetectionWorker i klasa CheckCustomerHistory pokazuju kako se AI radnici mogu besprekorno integrisati, unapređujući proces detekcije prevare inteligentnom analizom i sposobnostima donošenja odluka.

Analiza Sentimenta Kupaca

Evo još jednog sličnog primera kako možete implementirati radnika za analizu sentimenta kupaca. Ovog puta sa mnogo manje objašnjenja, pošto bi trebalo da već

razumete kako ovaj stil programiranja funkcioniše:

```
class CustomerSentimentAnalysisWorker
  include Wisper::Publisher

  def call(feedback)
    Result.ok(feedback)
      .and_then(PreprocessFeedback.method(:preprocess))
      .map(PerformSentimentAnalysis.method(:analyze))
      .map(ExtractKeyPhrases.method(:extract))
      .map(IdentifyTrends.method(:identify))
      .map(GenerateInsights.method(:generate)).then do |result|

      case result
      in { err: SentimentAnalysisError => error }
        Honeybadger.notify(error.message, context: {feedback:})
      in { ok: SentimentAnalysisResult => result }
        broadcast(:sentiment_analysis_completed, result)
      end
    end
  end
end
```

U ovom primeru, koraci CustomerSentimentAnalysisWorker-a uključuju predprocesiranje povratnih informacija (npr. uklanjanje šuma, tokenizaciju), izvođenje analize sentimenta kako bi se utvrdio opšti sentiment (pozitivan, negativan ili neutralan), izdvajanje ključnih fraza i tema, identifikovanje trendova i obrazaca, i generisanje primenljivih uvida na osnovu analize.

Primene u zdravstvu

U domenu zdravstva, AI radnici mogu pomoći medicinskim stručnjacima i istraživačima u različitim zadacima, što dovodi do poboljšanih ishoda lečenja i ubrzanih medicinskih otkrića. Neki primeri uključuju:

Prijem pacijenata

AI radnici mogu pojednostaviti proces prijema pacijenata automatizacijom različitih zadataka i pružanjem inteligentne pomoći.

Zakazivanje pregleda: AI radnici mogu upravljati zakazivanjem pregleda razumevanjem preferencija pacijenata, dostupnosti i hitnosti njihovih medicinskih potreba. Mogu komunicirati sa pacijentima kroz konverzacijske interfejse, vodeći ih kroz proces zakazivanja i pronalaženja najpogodnijih termina na osnovu zahteva pacijenta i dostupnosti zdravstvenog pružaoca usluga.

Prikupljanje medicinske istorije: Tokom prijema pacijenata, AI radnici mogu pomoći u prikupljanju i dokumentovanju medicinske istorije pacijenta. Mogu voditi interaktivne dijaloge sa pacijentima, postavljajući relevantna pitanja o njihovim prethodnim medicinskim stanjima, lekovima, alergijama i porodičnoj istoriji. AI radnici mogu koristiti tehnike obrade prirodnog jezika za tumačenje i strukturiranje prikupljenih informacija, osiguravajući da su tačno zabeležene u elektronskom zdravstvenom kartonu pacijenta.

Procena i stratifikacija simptoma: AI radnici mogu vršiti početne procene simptoma postavljanjem pitanja pacijentima o njihovim trenutnim simptomima, trajanju, ozbiljnosti i povezanim faktorima. Koristeći medicinske baze znanja i modele mašinskog učenja, ovi radnici mogu analizirati pružene informacije i generisati preliminarne diferencijalne dijagnoze ili preporučiti odgovarajuće sledeće korake, kao što je zakazivanje konsultacija sa zdravstvenim radnikom ili predlaganje mera samonege.

Verifikacija osiguranja: AI radnici mogu pomoći pri verifikaciji osiguranja tokom prijema pacijenata. Mogu prikupljati podatke o osiguranju pacijenta, komunicirati sa osiguravajućim društvima putem API-ja ili web servisa i proveravati podobnost za pokriće i beneficije. Ova automatizacija pomaže u pojednostavljivanju procesa verifikacije osiguranja, smanjujući administrativno opterećenje i osiguravajući tačno

prikupljanje informacija.

Edukacija pacijenata i uputstva: AI radnici mogu pružiti pacijentima relevantne edukativne materijale i uputstva zasnovana na njihovim specifičnim medicinskim stanjima ili predstojećim procedurama. Mogu isporučivati personalizovani sadržaj, odgovarati na česta pitanja i pružati smernice o pripremama pre pregleda, uputstvima za uzimanje lekova ili nezi nakon tretmana. Ovo pomaže da pacijenti budu informisani i angažovani tokom celokupnog zdravstvenog putovanja.

Korišćenjem AI radnika u prijemu pacijenata, zdravstvene organizacije mogu poboljšati efikasnost, smanjiti vreme čekanja i unaprediti celokupno iskustvo pacijenata. Ovi radnici mogu obavljati rutinske zadatke, prikupljati tačne informacije i pružati personalizovanu pomoć, omogućavajući zdravstvenim radnicima da se fokusiraju na pružanje kvalitetne nege pacijentima.

Procena rizika pacijenata

AI radnici mogu igrati ključnu ulogu u proceni rizika pacijenata analiziranjem različitih izvora podataka i primenom naprednih analitičkih tehnika.

Integracija podataka: AI radnici mogu prikupljati i razumeti podatke o pacijentima iz više izvora, kao što su elektronski zdravstveni kartoni (EZK), medicinski snimci, laboratorijski rezultati, nosivi uređaji i socijalne determinante zdravlja. Objedinjavanjem ovih informacija u sveobuhvatni profil pacijenta, AI radnici mogu pružiti holistički pregled zdravstvenog stanja pacijenta i faktora rizika.

Stratifikacija rizika: AI radnici mogu koristiti prediktivne modele za stratifikaciju pacijenata u različite kategorije rizika na osnovu njihovih individualnih karakteristika i zdravstvenih podataka. Ova stratifikacija rizika omogućava zdravstvenim radnicima da daju prioritet pacijentima kojima je potrebna neposrednija pažnja ili intervencija. Na primer, pacijenti identifikovani kao visokorizični za određeno stanje mogu biti označeni za pomnije praćenje, preventivne mere ili ranu intervenciju.

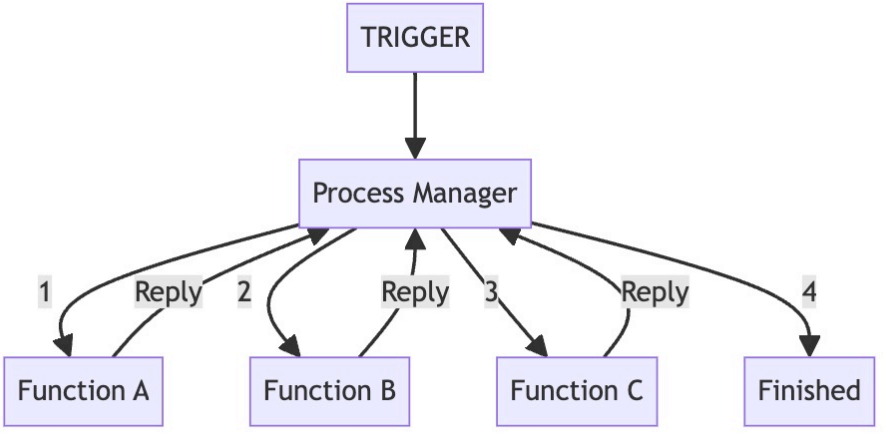
Personalizovani profili rizika: AI radnici mogu generisati personalizovane profile rizika za svakog pacijenta, ističući specifične faktore koji doprinose njihovim skorovima rizika. Ovi profili mogu uključivati uvide u životni stil pacijenta, genetske predispozicije, faktore okoline i socijalne determinante zdravlja. Pružanjem detaljnog pregleda faktora rizika, AI radnici mogu pomoći zdravstvenim radnicima da prilagode strategije prevencije i planove lečenja individualnim potrebama pacijenata.

Kontinuirano praćenje rizika: AI radnici mogu kontinuirano pratiti podatke o pacijentima i ažurirati procene rizika u realnom vremenu. Kako nove informacije postaju dostupne, kao što su promene u vitalnim znacima, laboratorijskim rezultatima ili pridržavanju terapije, AI radnici mogu preračunati skorove rizika i upozoriti zdravstvene radnike na sve značajne promene. Ovo proaktivno praćenje omogućava pravovremene intervencije i prilagođavanja planova nege pacijenata.

Podrška kliničkom odlučivanju: AI radnici mogu integrisati rezultate procene rizika u sisteme za podršku kliničkom odlučivanju, pružajući zdravstvenim radnicima preporuke i upozorenja zasnovana na dokazima. Na primer, ako skor rizika pacijenta za određeno stanje pređe određeni prag, AI radnik može podstaći zdravstvenog radnika da razmotri specifične dijagnostičke testove, preventivne mere ili opcije lečenja zasnovane na kliničkim smernicama i najboljoj praksi.

Ovi radnici mogu da obrade ogromne količine podataka o pacijentima, primene sofisticirane analize i generišu upotrebljive uvide koji podržavaju kliničko odlučivanje. Ovo na kraju dovodi do poboljšanih ishoda lečenja pacijenata, smanjenih troškova zdravstvene zaštite i unapređenog upravljanja zdravljem stanovništva.

AI radnik kao menadžer procesa



U kontekstu aplikacija vođenih veštačkom inteligencijom, radnik može biti dizajniran da funkcioniše kao Menadžer procesa, kako je opisano u knjizi "Enterprise Integration Patterns" autora Gregor Hohpe. Menadžer procesa je centralna komponenta koja održava stanje procesa i određuje sledeće korake obrade na osnovu međurezultata.

Kada AI radnik deluje kao Menadžer procesa, prima dolaznu poruku koja inicijalizuje proces, poznatu kao *poruka okidač*. AI radnik zatim održava stanje izvršavanja procesa (kao transkript konverzacije) i obrađuje poruku kroz niz koraka obrade implementiranih kao funkcije alata, koje mogu biti sekvencijalne ili paralelne, i pozivaju se po njegovom nahođenju.

Ako koristite klasu AI modela poput GPT-4 koja zna kako da izvršava funkcije paralelno, onda vaš radnik može istovremeno izvršavati više koraka. Priznajem, nisam to sam probao i moj instinkt govori da rezultati mogu varirati.

Nakon svakog pojedinačnog koraka obrade, kontrola se vraća nazad AI radniku, omogućavajući mu da odredi sledeći korak (ili korake) obrade na osnovu trenutnog stanja i dobijenih rezultata.

Čuvajte svoje poruke okidače

Prema mom iskustvu, pametno je implementirati poruku okidač kao objekat podržan bazom podataka. Na taj način je svaka instanca procesa identifikovana jedinstvenim primarnim ključem i daje vam mesto za čuvanje stanja povezanog sa izvršavanjem, uključujući AI transkript konverzacije.

Na primer, evo pojednostavljene verzije Olympia-inog modela AccountChange, koji predstavlja zahtev za promenu korisničkog naloga.

```
# == Schema Information
#
# Table name: account_changes
#
#  id          :uuid             not null, primary key
#  description :string
#  state       :string           not null
#  transcript  :jsonb
#  created_at  :datetime         not null
#  updated_at  :datetime         not null
#  account_id  :uuid             not null
#
# Indexes
#
#  index_account_changes_on_account_id  (account_id)
#
# Foreign Keys
#
#  fk_rails_...  (account_id => accounts.id)
#
class AccountChange < ApplicationRecord
  belongs_to :account

  validates :description, presence: true
```

```
25
26    after_commit -> {
27      broadcast(:account_change_requested, self)
28    }, on: :create
29
30    state_machine initial: :requested do
31      event :completed do
32        transition all => :complete
33      end
34      event :failed do
35        transition all => :requires_human_review
36      end
37    end
38  end
```

Klasa AccountChange služi kao poruka okidač koja pokreće proces za rukovanje zahtevom za promenu naloga. Primetite kako se emituje ka Olympia-inom Wisper podsistemu za objavljivanje/pretplatu nakon što se završi izvršavanje transakcije kreiranja.

Čuvanje poruke okidača u bazi podataka na ovaj način obezbeđuje trajni zapis svakog zahteva za promenu naloga. Svakoj instanci klase AccountChange dodeljuje se jedinstveni primarni ključ, što omogućava laku identifikaciju i praćenje pojedinačnih zahteva. Ovo je posebno korisno za potrebe revizorskog beleženja, jer omogućava sistemu da održava istorijski zapis svih promena naloga, uključujući kada su zatražene, koje promene su zatražene i trenutno stanje svakog zahteva.

U datom primeru, klasa AccountChange uključuje polja kao što su description za beleženje detalja zatražene promene, state za predstavljanje trenutnog stanja zahteva (npr. zatraženo, završeno, zahteva_ljudski_pregled), i transcript za čuvanje transkripta AI razgovora vezanog za zahtev. Polje description je stvarni prompt koji se koristi za pokretanje prvog chat completion-a sa AI-jem. Čuvanje ovih podataka pruža vredan kontekst i omogućava bolje praćenje i analizu procesa promene naloga.

Čuvanje poruka okidača u bazi podataka omogućava robusno rukovanje greškama i oporavak. Ako dođe do greške tokom obrade zahteva za promenu naloga, sistem

označava zahtev kao neuspešan i prelazi u stanje koje zahteva ljudsku intervenciju. Ovo osigurava da nijedan zahtev nije izgubljen ili zaboravljen, i da se svi problemi mogu pravilno rešiti.

AI radnik, kao Menadžer Procesa, obezbeđuje centralnu tačku kontrole i omogućava moćne mogućnosti izveštavanja i otklanjanja grešaka u procesu. Međutim, važno je napomenuti da korišćenje AI radnika kao Menadžera Procesa za svaki scenario toka rada u vašoj aplikaciji može biti preterano.

Integracija AI Radnika U Arhitekturu Vaše Aplikacije

Prilikom ugrađivanja AI radnika u arhitekturu vaše aplikacije, potrebno je razmotriti nekoliko tehničkih aspekata kako bi se osigurala glatka integracija i efikasna komunikacija između AI radnika i drugih komponenti aplikacije. Ovaj odeljak razmatra ključne aspekte dizajniranja tih interfejsa, rukovanja protokom podataka i upravljanja životnim ciklusom AI radnika.

Dizajniranje Jasnih Interfejsa i Komunikacionih Protokola

Za omogućavanje besprekorne integracije između AI radnika i drugih komponenti aplikacije, ključno je definisati jasne interfejse i komunikacione protokole. Razmotrite sledeće pristupe:

Integracija zasnovana na API-ju: Izložite funkcionalnost AI radnika kroz dobro definisane API-je, kao što su RESTful krajnje tačke ili GraphQL šeme. Ovo omogućava drugim komponentama da komuniciraju sa AI radnicima koristeći standardne HTTP zahteve i odgovore. Integracija zasnovana na API-ju pruža jasan ugovor između AI radnika i komponenti koje ih koriste, olakšavajući razvoj, testiranje i održavanje tačaka integracije.

Komunikacija zasnovana na porukama: Implementirajte obrasce komunikacije zasnovane na porukama, kao što su redovi poruka ili sistemi za objavljivanje-pretplatu, kako biste omogućili asinhronu interakciju između AI radnika i drugih komponenti. Ovaj pristup odvaja AI radnike od ostatka aplikacije, omogućavajući bolju skalabilnost, toleranciju na greške i labavo povezivanje. Komunikacija zasnovana na porukama je posebno korisna kada je obrada koju vrše AI radnici vremenski zahtevna ili resursno intenzivna, jer omogućava drugim delovima aplikacije da nastave sa izvršavanjem bez čekanja da AI radnici završe svoje zadatke.

Arhitektura vođena događajima: Dizajnirajte svoj sistem oko događaja i okidača koji aktiviraju AI radnike kada se ispune određeni uslovi. AI radnici se mogu pretplatiti na relevantne događaje i reagovati u skladu sa tim, izvršavajući svoje određene zadatke kada se događaji dese. Arhitektura vođena događajima omogućava obradu u realnom vremenu i dozvoljava da se AI radnici pozivaju po potrebi, smanjujući nepotrebnu potrošnju resursa. Ovaj pristup je pogodan za scenarije gde AI radnici treba da odgovore na specifične akcije ili promene u stanju aplikacije.

Rukovanje Protokom Podataka i Sinhronizacija

Prilikom integracije AI radnika u vašu aplikaciju, ključno je osigurati nesmetan protok podataka i održavati konzistentnost podataka između AI radnika i drugih komponenti. Razmotrite sledeće aspekte:

Priprema podataka: Pre nego što podatke prosledite AI radnicima, možda ćete morati da izvršite različite zadatke pripreme podataka, kao što su čišćenje, formatiranje i/ili transformacija ulaznih podataka. Ne samo da želite da osigurate da AI radnici mogu efikasno da obrađuju podatke, već i da se uverite da ne tračite tokene dajući pažnju informacijama koje radnik može smatrati beskorisnim u najboljem slučaju, a ometajućim u najgorem. Priprema podataka može uključivati zadatke poput uklanjanja šuma, rukovanja nedostajućim vrednostima ili konverzije tipova podataka.

Postojanost podataka: Kako ćete skladištiti i održavati podatke koji teku u i iz AI

radnika? Razmotrite faktore kao što su obim podataka, obrasci upita i skalabilnost. Da li je potrebno sačuvati transkript AI-ja kao odraz njegovog "procesa razmišljanja" za potrebe revizije ili otklanjanja grešaka, ili je dovoljno imati samo zapis rezultata?

Preuzimanje podataka: Dobavljanje podataka potrebnih radnicima može uključivati upite baza podataka, čitanje iz datoteka ili pristup eksternim API-jima. Pobrinite se da razmotrite latenciju i kako će AI radnici imati pristup najažurnijim podacima. Da li im je potreban potpun pristup vašoj bazi podataka ili biste trebali usko definisati opseg njihovog pristupa prema onome što rade? Šta je sa skaliranjem? Razmotrite mehanizme keširanja za poboljšanje performansi i smanjenje opterećenja na osnovne izvore podataka.

Sinhronizacija podataka: Kada više komponenti, uključujući AI radnike, pristupa i modifikuje deljene podatke, važno je implementirati odgovarajuće mehanizme sinhronizacije kako bi se održala konzistentnost podataka. Strategije zaključavanja baze podataka, kao što su optimističko ili pesimističko zaključavanje, mogu vam pomoći da sprečite konflikte i osigurate integritet podataka. Implementirajte tehnike upravljanja transakcijama za grupisanje povezanih operacija nad podacima i održavanje ACID svojstava (atomičnost, konzistentnost, izolacija i trajnost)

Upravljanje greškama i oporavak: Implementirajte robusne mehanizme za upravljanje greškama i oporavak kako biste se nosili sa problemima vezanim za podatke koji se mogu pojaviti tokom procesa protoka podataka. Elegantno upravljajte izuzecima i obezbedite smislene poruke o greškama koje pomažu pri otklanjanju grešaka. Implementirajte mehanizme ponovnih pokušaja i rezervne strategije za rukovanje privremenim otkazima ili prekidima mreže. Definišite jasne procedure za oporavak i vraćanje podataka u slučaju oštećenja ili gubitka podataka.

Pažljivim dizajniranjem i implementacijom mehanizama protoka i sinhronizacije podataka, možete osigurati da vaši AI radnici imaju pristup tačnim, konzistentnim i ažurnim podacima. Ovo im omogućava da efikasno obavljaju svoje zadatke i proizvode pouzdane rezultate.

Upravljanje životnim ciklusom AI radnika

Razvijte standardizovan proces za inicijalizaciju i konfiguraciju AI radnika. Naklonjen sam okvirima koji standardizuju način na koji definišete postavke kao što su imena modela, sistemske direktive i definicije funkcija. Osigurajte da je proces inicijalizacije automatizovan i ponovljiv kako bi se olakšalo raspoređivanje i skaliranje.

Implementirajte sveobuhvatne mehanizme za praćenje i beleženje kako biste pratili zdravlje i performanse AI radnika. Prikupljajte metrike kao što su iskorišćenost resursa, vreme obrade, stope grešaka i propusnost. Koristite centralizovane sisteme za beleženje kao što je ELK stack (Elasticsearch, Logstash, Kibana) za agregaciju i analizu zapisa iz više AI radnika.

Ugradite toleranciju na greške i otpornost u arhitekturu AI radnika. Implementirajte mehanizme za upravljanje greškama i oporavak kako biste elegantno rukovali otkazima ili izuzecima. Veliki jezički modeli su još uvek tehnologija u razvoju; pružaoci usluga često prestaju sa radom u neočekivanim trenucima. Koristite mehanizme ponovnih pokušaja i prekidače da sprečite kaskadne otkaze.

Kompozabilnost i orkestracija AI radnika

Jedna od ključnih prednosti arhitekture AI radnika je njena kompozabilnost, koja vam omogućava da kombinujete i orkestrate više AI radnika za rešavanje složenih problema. Razbijanjem većeg zadatka na manje, upravljivije podzadatke, kojima upravlja specijalizovani AI radnik, možete stvoriti moćne i fleksibilne sisteme. U ovom odeljku ćemo istražiti različite pristupe komponovanju i orkestraciji “mnoštva” AI radnika.

Ulančavanje AI radnika za višekoračne tokove rada

U mnogim scenarijima, složen zadatak se može razložiti na niz sekvencijalnih koraka, gde izlaz jednog AI radnika postaje ulaz za sledećeg. Ovo ulančavanje AI radnika

stvara višekoračni tok rada ili cevovod. Svaki AI radnik u lancu se fokusira na određeni podzadatak, a konačni izlaz je rezultat kombinovanih napora svih radnika.

Razmotrimo primer u kontekstu Ruby on Rails aplikacije za obradu korisnički generisanog sadržaja. Tok rada uključuje sledeće korake, koji su, priznajemo, verovatno pojedinačno previše jednostavni da bi vredelo razlagati ih na ovaj način u stvarnim slučajevima upotrebe, ali čine primer lakšim za razumevanje:

1. **Čišćenje teksta**: AI radnik zadužen za uklanjanje HTML oznaka, pretvaranje teksta u mala slova i upravljanje Unicode normalizacijom.

2. **Detekcija jezika**: AI radnik koji identifikuje jezik očišćenog teksta.

3. **Analiza sentimenta**: AI radnik koji određuje sentiment (pozitivan, negativan ili neutralan) teksta na osnovu detektovanog jezika.

4. **Kategorizacija sadržaja**: AI radnik koji klasifikuje tekst u predefinisane kategorije koristeći tehnike obrade prirodnog jezika.

Evo vrlo pojednostavljenog primera kako možete ulančati ove AI radnike koristeći Ruby:

```ruby
class ContentProcessor
  def initialize(text)
    @text = text
  end

  def process
    cleaned_text = TextCleanupWorker.new(@text).call
    language = LanguageDetectionWorker.new(cleaned_text).call
    sentiment = SentimentAnalysisWorker.new(cleaned_text, language).call
    category = CategorizationWorker.new(cleaned_text, language).call

    { cleaned_text:, language:, sentiment:, category: }
  end
end
```

U ovom primeru, klasa ContentProcessor se inicijalizuje sa sirovim tekstom i povezuje AI radnike zajedno u metodi process. Svaki AI radnik izvršava svoj specifični zadatak

i prosleđuje rezultat sledećem radniku u lancu. Konačni izlaz je heš koji sadrži očišćeni tekst, detektovani jezik, sentiment i kategoriju sadržaja.

Paralelna obrada za nezavisne AI radnike

U prethodnom primeru, AI radnici su povezani sekvencijalno, gde svaki radnik obrađuje tekst i prosleđuje rezultat sledećem radniku. Međutim, ako imate više AI radnika koji mogu nezavisno da rade na istom ulazu, možete optimizovati tok rada tako što ćete ih obraditi paralelno.

U datom scenariju, nakon što TextCleanupWorker izvrši čišćenje teksta, LanguageDetectionWorker, SentimentAnalysisWorker i CategorizationWorker mogu svi nezavisno da obrađuju očišćeni tekst. Pokretanjem ovih radnika paralelno, možete potencijalno smanjiti ukupno vreme obrade i poboljšati efikasnost vašeg toka rada.

Da biste postigli paralelnu obradu u Ruby-ju, možete iskoristiti tehnike konkurentnosti kao što su niti ili asinhrono programiranje. Evo primera kako možete modifikovati klasu ContentProcessor da obrađuje poslednja tri radnika paralelno koristeći niti:

```
require 'concurrent'

class ContentProcessor
  def initialize(text)
    @text = text
  end

  def process
    cleaned_text = TextCleanupWorker.new(@text).call

    language_future = Concurrent::Future.execute do
      LanguageDetectionWorker.new(cleaned_text).call
    end

    sentiment_future = Concurrent::Future.execute do
      SentimentAnalysisWorker.new(cleaned_text).call
```

```
    end

    category_future = Concurrent::Future.execute do
      CategorizationWorker.new(cleaned_text).call
    end

    language = language_future.value
    sentiment = sentiment_future.value
    category = category_future.value

    { cleaned_text:, language:, sentiment:, category: }
  end
end
```

U ovoj optimizovanoj verziji, koristimo biblioteku concurrent-ruby za kreiranje Concurrent::Future objekata za svakog od nezavisnih AI radnih procesa. Objekat Future predstavlja izračunavanje koje će se izvršiti asinhrono u zasebnoj niti.

Nakon koraka čišćenja teksta, kreiramo tri Future objekta: language_future, sentiment_future i category_future. Svaki Future izvršava svoj odgovarajući AI radni proces (LanguageDetectionWorker, SentimentAnalysisWorker i CategorizationWorker) u zasebnoj niti, prosleđujući cleaned_text kao ulaz.

Pozivanjem metode value na svakom Future objektu, čekamo da se izračunavanje završi i preuzimamo rezultat. Metoda value blokira izvršavanje dok rezultat ne bude dostupan, osiguravajući da su svi paralelni radni procesi završili obradu pre nastavka.

Na kraju, konstruišemo izlazni heš sa očišćenim tekstom i rezultatima iz paralelnih radnih procesa, baš kao u originalnom primeru.

Obradom nezavisnih AI radnih procesa paralelno, možete potencijalno smanjiti ukupno vreme obrade u poređenju sa sekvencijalnim izvršavanjem. Ova optimizacija je posebno korisna kada se radi sa vremenski zahtevnim zadacima ili kada se obrađuju velike količine podataka.

Međutim, važno je napomenuti da stvarni dobici u performansama zavise od različitih faktora, kao što su složenost svakog radnog procesa, dostupni sistemski resursi i režijski

troškovi upravljanja nitima. Uvek je dobra praksa testirati performanse i profilisati svoj kod kako biste odredili optimalni nivo paralelizma za vaš specifični slučaj upotrebe.

Dodatno, kada implementirate paralelnu obradu, vodite računa o svim deljenim resursima ili zavisnostima između radnih procesa. Osigurajte da radni procesi mogu raditi nezavisno bez konflikata ili stanja trke. Ako postoje zavisnosti ili deljeni resursi, možda ćete morati implementirati odgovarajuće mehanizme sinhronizacije kako biste održali integritet podataka i izbegli probleme poput međusobnog blokiranja ili nedoslednih rezultata.

Ruby-jev Global Interpreter Lock i asinhrona obrada

Važno je razumeti implikacije Ruby-jevog Global Interpreter Lock-a (GIL) kada razmatramo asinhronu obradu zasnovanu na nitima u Ruby-ju.

GIL je mehanizam u Ruby-jevom interpreteru koji osigurava da samo jedna nit može izvršavati Ruby kod u jednom trenutku, čak i na procesorima sa više jezgara. To znači da iako se više niti može kreirati i njima upravljati unutar Ruby procesa, samo jedna nit može aktivno izvršavati Ruby kod u bilo kom trenutku.

GIL je dizajniran da pojednostavi implementaciju Ruby interpretera i obezbedi sigurnost niti za Ruby-jeve interne strukture podataka. Međutim, on takođe ograničava mogućnost pravog paralelnog izvršavanja Ruby koda.

Kada koristite niti u Ruby-ju, kao što je to slučaj sa bibliotekom concurrent-ruby ili ugrađenom klasom Thread, niti su podložne ograničenjima GIL-a. GIL dozvoljava svakoj niti da izvršava Ruby kod tokom kratkog vremenskog isečka pre prebacivanja na drugu nit, stvarajući iluziju konkurentnog izvršavanja.

Međutim, zbog GIL-a, stvarno izvršavanje Ruby koda ostaje sekvencijalno. Dok jedna nit izvršava Ruby kod, druge niti su u suštini pauzirane, čekajući svoj red da dobiju GIL i izvrše se.

To znači da je asinhrona obrada zasnovana na nitima u Ruby-ju najefikasnija za U/I vezane zadatke, kao što je čekanje na odgovore eksternih API-ja (kao što su veliki jezički modeli hostovani od strane trećih strana) ili izvođenje U/I operacija sa fajlovima. Kada nit naiđe na U/I operaciju, može osloboditi GIL, omogućavajući drugim nitima da se izvršavaju dok čekaju da se U/I operacija završi.

S druge strane, za procesorski vezane zadatke, kao što su intenzivna izračunavanja ili dugotrajna obrada AI radnih procesa, GIL može ograničiti potencijalne dobitke u performansama kod paralelizma zasnovanog na nitima. Pošto samo jedna nit može izvršavati Ruby kod u jednom trenutku, ukupno vreme izvršavanja možda neće biti značajno smanjeno u poređenju sa sekvencijalnom obradom.

Da biste postigli pravo paralelno izvršavanje za procesorski vezane zadatke u Ruby-ju, možda ćete morati istražiti alternativne pristupe, kao što su:

- Korišćenje paralelizma zasnovanog na procesima sa više Ruby procesa, od kojih svaki radi na zasebnom procesorskom jezgru.
- Korišćenje eksternih biblioteka ili okvira koji pružaju nativna proširenja ili interfejse ka jezicima bez GIL-a, kao što su C ili Rust.,
- Korišćenje distribuiranih računarskih okvira ili redova poruka za distribuciju zadataka preko više mašina ili procesa.

Ključno je razmotriti prirodu vaših zadataka i ograničenja koja nameće GIL prilikom dizajniranja i implementacije asinhrone obrade u Ruby-ju. Dok asinhrona obrada zasnovana na nitima može pružiti prednosti za U/I vezane zadatke, možda neće ponuditi značajna poboljšanja performansi za procesorski vezane zadatke zbog ograničenja GIL-a.

Ensemble tehnike za poboljšanu preciznost

Ensemble tehnike podrazumevaju kombinovanje izlaza više AI radnih procesa kako bi se poboljšala ukupna preciznost ili robusnost sistema. Umesto oslanjanja na jedan AI radni proces, ensemble tehnike koriste kolektivnu inteligenciju više radnih procesa za donošenje informisanijih odluka.

Ansambli su posebno važni ako različiti delovi vašeg radnog toka najbolje funkcionišu sa različitim AI modelima, što je češća pojava nego što možda mislite. Moćni modeli poput GPT-4 su izuzetno skupi u poređenju sa manje sposobnim opcijama otvorenog koda, i verovatno nisu potrebni za svaki pojedinačni korak radnog toka vaše aplikacije.

Jedna uobičajena tehnika ansambla je većinsko glasanje, gde više AI izvršilaca nezavisno obrađuje isti ulaz, a konačni izlaz se određuje većinskim konsenzusom. Ovaj pristup može pomoći u ublažavanju uticaja grešaka pojedinačnih izvršilaca i poboljšati ukupnu pouzdanost sistema.

Razmotrimo primer gde imamo tri AI izvršioca za analizu sentimenta, od kojih svaki koristi različit model ili je snabdeven različitim kontekstom. Možemo kombinovati njihove izlaze koristeći većinsko glasanje kako bismo odredili konačno predviđanje sentimenta.

```ruby
class SentimentAnalysisEnsemble
  def initialize(text)
    @text = text
  end

  def analyze
    predictions = [
      SentimentAnalysisWorker1.new(@text).analyze,
      SentimentAnalysisWorker2.new(@text).analyze,
      SentimentAnalysisWorker3.new(@text).analyze
    ]

    predictions
      .group_by { |sentiment| sentiment }
      .max_by { |_, votes| votes.size }
      .first

  end
end
```

U ovom primeru, klasa SentimentAnalysisEnsemble. se inicijalizuje sa tekstom i poziva tri različita AI radnika za analizu sentimenta. Metoda analyze prikuplja predviđanja od svakog radnika i određuje većinski sentiment koristeći metode group_by i max_by. Konačni rezultat je sentiment koji dobija najviše glasova od ansambla radnika

Ansambli su očigledno slučaj gde eksperimentisanje sa paralelizmom može biti vredno vašeg vremena.

Dinamički odabir i pozivanje AI radnika

U nekim, ako ne i u većini slučajeva, određeni AI radnik koji će biti pozvan može zavisiti od uslova izvršavanja ili korisničkih unosa. Dinamički odabir i pozivanje AI radnika omogućavaju fleksibilnost i prilagodljivost sistema.

Možda ćete biti u iskušenju da pokušate da uklopite mnogo funkcionalnosti u jednog AI radnika, dajući mu mnoštvo funkcija i veliki komplikovani prompt koji objašnjava kako ih pozvati. Odupreti se iskušenju, verujte mi. Jedan od razloga zašto se pristup o kojem govorimo u ovom poglavlju zove "Mnoštvo radnika" je da nas podseti da je poželjno imati mnogo specijalizovanih radnika, od kojih svaki obavlja svoj mali posao u službi veće svrhe.

Na primer, razmotrite četbot aplikaciju gde su različiti AI radnici odgovorni za obradu različitih tipova korisničkih upita. Na osnovu korisničkog unosa, aplikacija dinamički bira odgovarajućeg AI radnika za obradu upita.

```
class ChatbotController < ApplicationController
  def process_query
    query = params[:query]
    query_type = QueryClassifierWorker.new(query).classify

    case query_type
    when 'greeting'
      response = GreetingWorker.new(query).generate_response
    when 'product_inquiry'
      response = ProductInquiryWorker.new(query).generate_response
    when 'order_status'
      response = OrderStatusWorker.new(query).generate_response
    else
      response = DefaultResponseWorker.new(query).generate_response
    end

    render json: { response: response }
  end
end
```

U ovom primeru, ChatbotController prima korisnički upit kroz process_query akciju. Prvo koristi QueryClassifierWorker da odredi tip upita. Na osnovu klasifikovanog tipa upita, kontroler dinamički bira odgovarajućeg AI radnika za generisanje odgovora. Ova dinamička selekcija omogućava četbotu da obrađuje različite tipove upita i usmerava ih ka relevantnim AI radnicima.

Pošto je rad QueryClassifierWorker-a relativno jednostavan i ne zahteva mnogo konteksta ili definicija funkcija, verovatno ga možete implementirati koristeći ultra-brz mali VJM kao što je mistralai/mixtral-8x7b-instruct:nitro. On ima mogućnosti koje su bliske GPT-4 nivou na mnogim zadacima i, u trenutku dok ovo pišem, Groq ga može posluživati neverovatnom brzinom od 444 tokena u sekundi.

Kombinovanje tradicionalne OPJ sa VJM-ovima

Iako su Veliki jezički modeli (VJM) revolucionirali oblast obrade prirodnog jezika (OPJ), nudeći neprevaziđenu svestranost i performanse u širokom spektru zadataka, oni nisu uvek najefikasnije ili najisplativije rešenje za svaki problem. U mnogim slučajevima, kombinovanje tradicionalnih OPJ tehnika sa VJM-ovima može dovesti do optimizovanijih, ciljanijih i ekonomičnijih pristupa rešavanju specifičnih OPJ izazova.

Zamislite VJM-ove kao švajcarske noževe OPJ-a--neverovatno svestrane i moćne, ali ne nužno najbolji alat za svaki posao. Ponekad, namenski alat poput vadičepa ili otvarača za konzerve može biti efikasniji za određeni zadatak. Slično tome, tradicionalne OPJ tehnike, kao što su grupisanje dokumenata, identifikacija tema i klasifikacija, često mogu pružiti ciljanije i ekonomičnije rešenje za određene aspekte vašeg OPJ procesa.

Jedna od ključnih prednosti tradicionalnih OPJ tehnika je njihova računarska efikasnost. Ove metode, koje se često oslanjaju na jednostavnije statističke modele ili pristupe zasnovane na pravilima, mogu obrađivati velike količine tekstualnih podataka mnogo brže i sa manjim računarskim opterećenjem u poređenju sa VJM-ovima. Ovo ih čini posebno pogodnim za zadatke koji uključuju analizu i organizaciju velikih korpusa dokumenata, kao što je grupisanje sličnih članaka ili identifikacija ključnih tema unutar kolekcije tekstova.

Štaviše, tradicionalne OPJ tehnike često mogu postići visoku tačnost i preciznost za specifične zadatke, posebno kada su obučene na domenski specifičnim skupovima

podataka. Na primer, dobro podešen klasifikator dokumenata koji koristi tradicionalne algoritme mašinskog učenja poput Mašina sa vektorima podrške (MVP) ili Naivnog Bajesa može precizno kategorisati dokumente u predefinisane kategorije uz minimalne računarske troškove.

Međutim, VJM-ovi zaista blistaju kada su u pitanju zadaci koji zahtevaju dublje razumevanje jezika, konteksta i rezonovanja. Njihova sposobnost da generišu koherentan i kontekstualno relevantan tekst, odgovaraju na pitanja i sumiraju duge pasuse je neprevaziđena tradicionalnim OPJ metodama. VJM-ovi mogu efikasno da se nose sa složenim lingvističkim fenomenima, kao što su dvosmislenost, koreferencija i idiomatski izrazi, čineći ih neprocenjivim za zadatke koji zahtevaju generisanje ili razumevanje prirodnog jezika.

Prava snaga leži u kombinovanju tradicionalnih OPJ tehnika sa VJM-ovima kako bi se stvorili hibridni pristupi koji koriste prednosti oba. Korišćenjem tradicionalnih OPJ metoda za zadatke poput pretprocesiranja dokumenata, grupisanja i ekstrakcije tema, možete efikasno organizovati i strukturirati vaše tekstualne podatke. Ove strukturirane informacije se zatim mogu proslediti VJM-ovima za naprednije zadatke, kao što su generisanje rezimea, odgovaranje na pitanja ili kreiranje sveobuhvatnih izveštaja.

Na primer, razmotrimo slučaj upotrebe gde želite da generišete izveštaj o trendovima za specifičan domen na osnovu velikog korpusa pojedinačnih dokumenata o trendovima. Umesto da se oslanjate isključivo na VJM-ove, što može biti računarski skupo i vremenski zahtevno za obradu velikih količina teksta, možete primeniti hibridni pristup:

1. Koristite tradicionalne OPJ tehnike, kao što su modelovanje tema (npr. Latentna Dirihleova alokacija) ili algoritmi grupisanja (npr. K-sredine), za grupisanje sličnih dokumenata o trendovima i identifikaciju ključnih tema unutar korpusa.
2. Prosledite grupisane dokumente i identifikovane teme VJM-u, koristeći njegove superiorne sposobnosti razumevanja i generisanja jezika za kreiranje koherentnih i informativnih rezimea za svaku grupu ili temu.

3. Na kraju, koristite VJM za generisanje sveobuhvatnog izveštaja o trendovima kombinovanjem pojedinačnih rezimea, isticanjem najznačajnijih trendova i pružanjem uvida i preporuka na osnovu objedinjenih informacija.

Kombinovanjem tradicionalnih OPJ tehnika sa VJM-ovima na ovaj način, možete efikasno obraditi velike količine tekstualnih podataka, izvući smislene uvide i generisati visokokvalitetne izveštaje uz optimizaciju računarskih resursa i troškova.

Kada se upuštate u NLP projekte, neophodno je pažljivo proceniti specifične zahteve i ograničenja svakog zadatka i razmotriti kako se tradicionalne NLP metode i VJM mogu zajedno iskoristiti za postizanje najboljih rezultata. Kombinovanjem efikasnosti i preciznosti tradicionalnih tehnika sa svestranošću i snagom VJM-a, možete kreirati izuzetno efikasna i ekonomična NLP rešenja koja donose vrednost vašim korisnicima i zainteresovanim stranama.

Upotreba alata

U domenu razvoja aplikacija vođenih veštačkom inteligencijom, koncept “upotrebe alata” ili “pozivanja funkcija” pojavio se kao moćna tehnika koja omogućava vašem LLM-u da se poveže sa spoljnim alatima, API-jima, funkcijama, bazama podataka i drugim resursima. Ovaj pristup omogućava bogatiji skup ponašanja od samog generisanja teksta, kao i dinamičnije interakcije između vaših AI komponenti i ostatka ekosistema vaše aplikacije. Kao što ćemo razmotriti u ovom poglavlju, upotreba alata vam takođe pruža mogućnost da vaš AI model generiše podatke na strukturiran način.

Šta je upotreba alata?

Upotreba alata, takođe poznata kao pozivanje funkcija, je tehnika koja omogućava programerima da definišu listu funkcija sa kojima LLM može da komunicira tokom

procesa generisanja. Ovi alati mogu biti sve od jednostavnih pomoćnih funkcija do složenih API-ja ili upita baze podataka. Omogućavajući LLM-u pristup ovim alatima, programeri mogu proširiti mogućnosti modela i omogućiti mu da izvršava zadatke koji zahtevaju spoljno znanje ili akcije.

Slika 8. Primer definicije funkcije za AI radnika koji analizira dokumente

```
FUNCTION = {
  name: "save_analysis",
  description: "Save analysis data for document",
  parameters: {
    type: "object",
    properties: {
      title: {
        type: "string",
        maxLength: 140
      },
      summary: {
        type: "string",
        description: "comprehensive multi-paragraph summary with
                      overview and list of sections (if applicable)"
      },
      tags: {
        type: "array",
        items: {
          type: "string",
          description: "lowercase tags representing main themes
                        of the document"
        }
      }
    },
    "required": %w[title summary tags]
  }
}.freeze
```

Ključna ideja iza upotrebe alata je da se LLM-u omogući dinamički odabir i izvršavanje odgovarajućih alata na osnovu korisničkog unosa ili zadatka koji treba obaviti. Umesto da se oslanja isključivo na prethodno obučeno znanje modela, upotreba alata omogućava LLM-u da koristi eksterne resurse za generisanje preciznijih, relevantnijih

i upotrebljivijih odgovora. Upotreba alata čini tehnike kao što je RAG (Generisanje potpomognuto pretraživanjem) mnogo lakšim za implementaciju nego što bi to inače bilo.

> Imajte na umu da, osim ako nije drugačije navedeno, ova knjiga pretpostavlja da vaš AI model nema pristup ugrađenim alatima na strani servera. Sve alate koje želite da učinite dostupnim vašem AI-ju morate eksplicitno deklarisati u svakom API zahtevu, sa odredbama za njihovo izvršavanje ako i kada vaš AI naznači da bi želeo da koristi taj alat u svom odgovoru.

Potencijal upotrebe alata

Upotreba alata otvara širok spektar mogućnosti za aplikacije zasnovane na veštačkoj inteligenciji. Evo nekoliko primera šta se može postići upotrebom alata:

1. **Četbotovi i virtuelni asistenti**: Povezivanjem LLM-a sa eksternim alatima, četbotovi i virtuelni asistenti mogu obavljati složenije zadatke, kao što su preuzimanje informacija iz baza podataka, izvršavanje API poziva ili interakcija sa drugim sistemima. Na primer, četbot bi mogao koristiti CRM alat za promenu statusa ponude na osnovu korisničkog zahteva.
2. **Analiza podataka i uvidi**: LLM-ovi se mogu povezati sa alatima ili bibliotekama za analizu podataka kako bi izvršavali napredne zadatke obrade podataka. Ovo omogućava aplikacijama da generišu uvide, sprovode uporedne analize ili pružaju preporuke zasnovane na podacima na osnovu korisničkih upita.
3. **Pretraga i preuzimanje informacija**: Upotreba alata omogućava LLM-ovima interakciju sa pretraživačima, vektorskim bazama podataka ili drugim sistemima za preuzimanje informacija. Transformisanjem korisničkih upita u upite za

pretragu, LLM može preuzeti relevantne informacije iz više izvora i pružiti sveobuhvatne odgovore na korisnička pitanja.

4. **Integracija sa eksternim servisima**: Upotreba alata omogućava besprekornu integraciju između aplikacija zasnovanih na veštačkoj inteligenciji i eksternih servisa ili API-ja. Na primer, LLM bi mogao da komunicira sa API-jem za vremenske prilike kako bi pružio ažurne informacije o vremenu ili sa API-jem za prevođenje kako bi generisao višejezične odgovore.

Tok rada sa alatima

Tok rada sa alatima obično uključuje četiri ključna koraka:

1. Uključivanje definicija funkcija u kontekst zahteva
2. Dinamički (ili eksplicitni) odabir alata
3. Izvršavanje funkcije/a
4. Opcionalni nastavak originalnog upita

Hajde da detaljno pregledamo svaki od ovih koraka.

Uključivanje definicija funkcija u kontekst zahteva

AI zna koje alate ima na raspolaganju jer mu dajete listu kao deo vašeg zahteva za završetak (obično definisano kao funkcije koristeći varijantu JSON šeme).

Precizna sintaksa definicije alata je specifična za model.

Ovo je način kako se definiše funkcija get_weather u Claude 3:

```
{
    "name": "get_weather",
    "description": "Get the current weather in a given location",
    "input_schema": {
        "type": "object",
        "properties": {
            "location": {
                "type": "string",
                "description": "The city and state, e.g. San Francisco, CA"
            },
            "unit": {
                "type": "string",
                "enum": ["celsius", "fahrenheit"],
                "description": "The unit of temperature"
        }
    },
    "required": ["location"]
    }
}
```

A ovo je način kako biste definisali istu funkciju za GPT-4, prosleđujući je kao vrednost parametra tools:

```
{
    "name": "get_current_weather",
    "description": "Get the current weather in a given location",
    "parameters": {
        "type": "object",
        "properties": {
            "location": {
                "type": "string",
                "description": "The city and state, e.g. San Francisco, CA",
            },
            "unit": {
                "type": "string",
                "enum": ["celsius", "fahrenheit"],
                "description": "The unit of temperature"
            },
        },
        "required": ["location"],
```

```
    },
}
```

> Skoro isto, samo drugačije bez očiglednog razloga! Kako iritantno.

Definicije funkcija određuju naziv, opis i ulazne parametre. Ulazni parametri se mogu dodatno definisati korišćenjem atributa kao što su enumeracije za ograničavanje prihvatljivih vrednosti, i određivanjem da li je parametar obavezan ili ne.

Pored samih definicija funkcija, možete takođe uključiti uputstva ili kontekst o tome zašto i kako koristiti funkciju u sistemskoj direktivi.

Na primer, moj alat za pretragu veba u Olympia uključuje ovu sistemsku direktivu, koja podseća VI da ima pomenute alate na raspolaganju:

```
The `google_search` and `realtime_search` functions let you do research
on behalf of the user. In contrast to Google, realtime search is powered
by Perplexity and provides real-time information to curated current events
databases and news sources. Make sure to include URLs in your response so
user can do followup research.
```

Pružanje detaljnih opisa smatra se najvažnijim faktorom u performansama alata. Vaši opisi bi trebalo da objasne svaki detalj o alatu, uključujući:

- Šta alat radi
- Kada bi trebalo da se koristi (a kada ne bi trebalo)
- Šta svaki parametar znači i kako utiče na ponašanje alata
- Sve važne napomene ili ograničenja koja se odnose na implementaciju alata

Što više konteksta možete dati veštačkoj inteligenciji o vašim alatima, to će ona biti bolja u odlučivanju kada i kako da ih koristi. Na primer, Anthropic preporučuje najmanje 3-4 rečenice po opisu alata za svoju Claude 3 seriju, više ako je alat složen.

Nije nužno intuitivno, ali opisi se također smatraju važnijim od primera. Iako možete uključiti primere kako se koristi alat u njegovom opisu ili u pratećem promptu, to je manje važno od jasnog i sveobuhvatnog objašnjenja svrhe i parametara alata. Primere dodajte tek nakon što ste u potpunosti razradili opis.

Evo primera Stripe-ovske API funkcijske specifikacije:

```
{
  "name": "createPayment",
  "description": "Create a new payment request",
  "parameters": {
    "type": "object",
    "properties": {
      "transaction_amount": {
        "type": "number",
        "description": "The amount to be paid"
      },
      "description": {
        "type": "string",
        "description": "A brief description of the payment"
      },
      "payment_method_id": {
        "type": "string",
        "description": "The payment method to be used"
      },
      "payer": {
        "type": "object",
        "description": "Information about the payer, including their name,
                        email, and identification number",
        "properties": {
          "name": {
            "type": "string",
            "description": "The payer's name"
        },
        "email": {
          "type": "string",
          "description": "The payer's email address"
        },
        "identification": {
          "type": "object",
          "description": "The payer's identification number",
```

```
            "properties": {
              "type": {
                "type": "string",
                "description": "Identification document (e.g. CPF, CNPJ)"
              },
              "number": {
                "type": "string",
                "description": "The identification number"
              }
            },
            "required": [ "type", "number" ]
          }
        },
        "required": [ "name", "email", "identification" ]
      }
    }
}
```

U praksi, neki modeli imaju poteškoća sa ugneždenim funkcijskim specifikacijama i sa složenim tipovima izlaznih podataka kao što su nizovi, rečnici itd. Ali teoretski, trebalo bi da možete da dostavite JSON Schema specifikacije proizvoljne dubine!

Dinamički odabir alata

Kada izvršavate chat completion koji uključuje definicije alata, LLM dinamički bira najadekvatniji alat(e) za upotrebu i generiše potrebne ulazne parametre za svaki alat.

U praksi, sposobnost AI-ja da pozove *tačno* pravu funkciju i *tačno* prati vašu specifikaciju za ulazne podatke je promenljiva. Postavljanje temperature hiperparametra skroz na 0.0 dosta pomaže, ali prema mom iskustvu i dalje ćete povremeno dobijati greške. Te greške uključuju halucirane nazive funkcija, pogrešno imenovane ili jednostavno nedostajuće ulazne parametre. Parametri se prosleđuju kao JSON, što znači da ćete ponekad videti greške uzrokovane odsečenim, pogrešno navedenim ili na drugi način neispravnim JSON-om.

Obrasci Samozalečenja podataka mogu pomoći da se automatski poprave pozivi funkcija koji se prekidaju zbog sintaksnih grešaka.

Prinudni (odnosno Eksplicitni) odabir alata

Neki modeli vam daju opciju da nametnete pozivanje određene funkcije kao parametar u zahtevu. U suprotnom, da li će pozvati funkciju ili ne u potpunosti zavisi od procene AI-ja.

Mogućnost prinudnog pozivanja funkcije je ključna u određenim scenarijima gde želite da osigurate da se određeni alat ili funkcija izvrši, bez obzira na proces dinamičkog odabira AI-ja. Postoji nekoliko razloga zašto je ova mogućnost važna:

1. **Eksplicitna kontrola**: Možda koristite AI kao *Diskretnu komponentu* ili u unapred definisanom toku rada koji zahteva izvršavanje određene funkcije u određeno vreme. Prinudnim pozivanjem možete garantovati da će željena funkcija biti pozvana umesto da ljubazno molite AI da to uradi.
2. **Debagovanje i testiranje**: Prilikom razvoja i testiranja aplikacija vođenih AI-jem, mogućnost prinudnog pozivanja funkcija je neprocenjiva za potrebe debagovanja. Eksplicitnim pokretanjem određenih funkcija možete izolovati i testirati pojedinačne komponente vaše aplikacije. Ovo vam omogućava da proverite ispravnost implementacija funkcija, validirate ulazne parametre i osigurate da se vraćaju očekivani rezultati.
3. **Rukovanje ivičnim slučajevima**: Mogu postojati ivični slučajevi ili izuzetni scenariji gde proces dinamičkog odabira AI-ja možda neće odabrati izvršavanje funkcije koju bi trebalo, a vi to znate na osnovu spoljnih procesa. U takvim slučajevima, mogućnost prinudnog pozivanja funkcije vam omogućava da eksplicitno rukujete ovim situacijama. Definišite pravila ili uslove u logici vaše aplikacije da odredite kada da premostite diskreciju AI-ja.

4. **Konzistentnost i reproduktivnost**: Ako imate određeni niz funkcija koje treba izvršiti određenim redosledom, prinudno pozivanje garantuje da će se isti redosled pratiti svaki put. Ovo je posebno važno u aplikacijama gde su konzistentnost i predvidljivo ponašanje kritični, kao što su finansijski sistemi ili naučne simulacije.
5. **Optimizacija performansi**: U nekim slučajevima, prinudno pozivanje funkcije može dovesti do optimizacije performansi. Ako znate da je određena funkcija potrebna za određeni zadatak i da bi proces dinamičkog odabira AI-ja mogao uvesti nepotreban overhead, možete zaobići proces selekcije i direktno pozvati potrebnu funkciju. Ovo može pomoći u smanjenju latencije i poboljšanju ukupne efikasnosti vaše aplikacije.

Ukratko, mogućnost prinudnog pozivanja funkcija u aplikacijama vođenim AI-jem pruža eksplicitnu kontrolu, pomaže u debagovanju i testiranju, rukuje ivičnim slučajevima, osigurava konzistentnost i reproduktivnost. To je moćan alat u vašem arsenalu, ali moramo razgovarati o još jednom aspektu ove važne funkcionalnosti.

U mnogim slučajevima donošenja odluka, uvek želimo da model izvrši poziv funkcije i možda nikada ne želimo da model odgovori samo svojim internim znanjem. Na primer, ako usmeravate između više modela specijalizovanih za različite zadatke (višejezični unos, matematika itd.), možete koristiti model za pozivanje funkcija da delegira zahteve jednom od pomoćnih modela i nikada ne odgovara samostalno.

Parametar izbora alata

GPT-4 i drugi jezički modeli koji podržavaju pozivanje funkcija daju vam parametar tool_choice za kontrolu da li je upotreba alata obavezna kao deo završetka. Ovaj parametar ima tri moguće vrednosti:

- auto daje AI-ju punu diskreciju oko korišćenja alata ili jednostavnog odgovaranja

- required govori AI-ju da *mora* pozvati alat *umesto* da odgovori, ali ostavlja izbor alata AI-ju
- Treća opcija je postavljanje parametra name_of_function koji želite da prisilno pozovete. Više o tome u sledećem odeljku.

Imajte na umu da ako postavite izbor alata na required, model će biti primoran da izabere najrelevantniju funkciju za poziv od onih koje su mu dostupne, čak i ako nijedna zaista ne odgovara upitu. U trenutku objavljivanja, nije mi poznato da postoji model koji će vratiti prazan tool_calls odgovor ili na neki drugi način dati do znanja da nije pronašao odgovarajuću funkciju za poziv.

Forsiranje funkcije za dobijanje strukturiranog izlaza

Mogućnost forsiranja poziva funkcije vam daje način da dobijete strukturirane podatke iz chat završetka umesto da ih sami morate izvlačiti iz tekstualnog odgovora.

> Zašto je forsiranje funkcija za dobijanje strukturiranog izlaza toliko važno? Jednostavno rečeno, zato što je izvlačenje strukturiranih podataka iz izlaza VJM-a pravi glavobolja. Možete sebi malo olakšati život tako što ćete tražiti podatke u XML-u, ali onda morate parsirati XML. I šta radite kada taj XML nedostaje jer je vaša VI odgovorila: “Žao mi je, ali ne mogu da generišem podatke koje ste tražili zato što bla, bla, bla…”

Kada koristite alate na ovaj način:

- Verovatno bi trebalo da definišete jedan alat u vašem zahtevu

- Ne zaboravite da forsirate korišćenje njegove funkcije pomoću tool_choice parametra
- Zapamtite da će model proslediti unos alatu, tako da naziv alata i opis treba da budu iz perspektive modela, a ne vaše

Ova poslednja tačka zaslužuje primer radi jasnoće. Recimo da tražite od VI da uradi analizu sentimenta korisničkog teksta. Naziv funkcije ne bi bio analyze_sentiment, već nešto poput save_sentiment_analysis. VI je ta koja radi analizu sentimenta, *ne alat*. Sve što alat radi (iz perspektive VI) je čuvanje rezultata analize.

Evo primera korišćenja Claude 3 za beleženje rezimea slike u dobro strukturirani JSON, ovog puta iz komandne linije koristeći curl:

```
curl https://api.anthropic.com/v1/messages \
     --header "content-type: application/json" \
     --header "x-api-key: $ANTHROPIC_API_KEY" \
     --header "anthropic-version: 2023-06-01" \
     --header "anthropic-beta: tools-2024-04-04" \
     --data \
'{
    "model": "claude-3-sonnet-20240229",
    "max_tokens": 1024,
    "tools": [{
        "name": "record_summary",
        "description": "Record summary of image into well-structured JSON.",
        "input_schema": {
            "type": "object",
            "properties": {
                "key_colors": {
                    "type": "array",
                    "items": {
                        "type": "object",
                        "properties": {
                            "r": {
                                "type": "number",
                                "description": "red value [0.0, 1.0]"
                            },
                            "g": {
```

```
                                "type": "number",
                                "description": "green value [0.0, 1.0]"
                            },
                            "b": {
                                "type": "number",
                                "description": "blue value [0.0, 1.0]"
                            },
                            "name": {
                                "type": "string",
                                "description": "Human-readable color name
                                                in snake_case, e.g.
                                                \"olive_green\"or
                                                \"turquoise\""
                            }
                        },
                        "required": [ "r", "g", "b", "name" ]
                    },
                    "description": "Key colors in the image. Four or less."
                },
                "description": {
                    "type": "string",
                    "description": "Image description. 1-2 sentences max."
                },
                "estimated_year": {
                    "type": "integer",
                    "description": "Estimated year that the image was taken,
                                    if is it a photo. Only set this if the
                                    image appears to be non-fictional.
                                    Rough estimates are okay!"
                }
            },
            "required": [ "key_colors", "description" ]
        }
    }],
    "messages": [
        {
            "role": "user",
            "content": [
                {
                    "type": "image",
                    "source": {
                        "type": "base64",
```

```
                        "media_type": "'$IMAGE_MEDIA_TYPE'",
                        "data": "'$IMAGE_BASE64'"
                    }
                },
                {
                    "type": "text",
                    "text": "Use `record_summary` to describe this image."
                }
            ]
        }
    ]
}'
```

U datom primeru, koristimo Claude 3 model kompanije Anthropic za generisanje strukturiranog JSON rezimea slike. Evo kako to funkcioniše:

1. Definišemo jedan alat pod nazivom record_summary u nizu tools u sadržaju zahteva. Ovaj alat je zadužen za beleženje rezimea slike u dobro strukturiranom JSON formatu.
2. Alat record_summary ima input_schema koji određuje očekivanu strukturu JSON izlaza. On definiše tri svojstva:
    - key_colors: Niz objekata koji predstavljaju ključne boje u slici. Svaki objekat boje ima svojstva za crvenu, zelenu i plavu vrednost (u opsegu od 0.0 do 1.0) i ljudski čitljiv naziv boje u snake_case formatu.
    - description: String svojstvo za kratak opis slike, ograničen na 1-2 rečenice.
    - estimated_year: Opciono celobrojno svojstvo za procenjenu godinu kada je slika snimljena, ako izgleda da je u pitanju stvarna fotografija.
3. U nizu messages, dostavljamo podatke slike kao base64-kodirani string zajedno sa tipom medija. Ovo omogućava modelu da obradi sliku kao deo ulaza.
4. Takođe upućujemo Claude-a da koristi alat record_summary za opisivanje slike.
5. Kada se zahtev pošalje Claude 3 modelu, on analizira sliku i generiše JSON rezime na osnovu određenog input_schema. Model izdvaja ključne boje, pruža kratak opis i procenjuje godinu kada je slika snimljena (ako je primenljivo).

6. Generisani JSON rezime se prosleđuje kao parametar alatu record_summary, pružajući strukturirani prikaz ključnih karakteristika slike.

Korišćenjem alata record_summary sa dobro definisanim input_schema, možemo dobiti strukturirani JSON rezime slike bez oslanjanja na ekstrakciju običnog teksta. Ovaj pristup osigurava da izlaz prati dosledan format i može se lako parsirati i obraditi od strane nizvodnih komponenti aplikacije.

Mogućnost forsiranja poziva funkcije i određivanja očekivane izlazne strukture je moćna karakteristika upotrebe alata u aplikacijama vođenim veštačkom inteligencijom. To omogućava programerima da imaju više kontrole nad generisanim izlazom i pojednostavljuje integraciju podataka generisanih veštačkom inteligencijom u tok rada njihove aplikacije.

Izvršavanje funkcije(a)

Definisali ste funkcije i uputili svoj AI, koji je odlučio da treba da pozove jednu od vaših funkcija. Sada je vreme da vaš aplikacioni kod ili biblioteka, ako koristite Ruby gem poput raix-rails, prosledi poziv funkcije i njene parametre odgovarajućoj implementaciji *u vašem aplikacionom kodu.*

Vaš aplikacioni kod odlučuje šta da radi sa rezultatima izvršavanja funkcije. Možda to podrazumeva jednu liniju koda u lambda izrazu, ili možda podrazumeva pozivanje eksternog API-ja. Možda uključuje pozivanje druge AI komponente, ili možda uključuje stotine ili čak hiljade linija koda u ostatku vašeg sistema. To je u potpunosti do vas.

Ponekad je poziv funkcije kraj operacije, ali ako rezultati predstavljaju informacije u lancu razmišljanja koji AI treba da nastavi, tada vaš aplikacioni kod mora da ubaci rezultate izvršavanja u transkript razgovora i dozvoli AI-ju da nastavi obradu.

Na primer, evo Raix deklaracije funkcije koju koristi Olympia-in AccountManager za komunikaciju sa našim klijentima kao deo Inteligentne orkestracije toka rada za korisničku podršku.

```ruby
class AccountManager
  include Raix::ChatCompletion
  include Raix::FunctionDispatch

  # lots of other functions...

  function :notify_account_owner,
    "Don't share UUID. Mention dollars if subscription changed",
    message: { type: "string" } do |arguments|
      account.owner.freeform_notify(
        subject: "Account Change Notification",
        message: arguments[:message]
      )
      "Notified account owner"
    end
```

Možda nije odmah jasno šta se ovde dešava, pa ću to razložiti.

1. Klasa AccountManager definiše mnoge funkcije vezane za upravljanje nalozima. Može da promeni vaš plan, dodaje i uklanja članove tima, između ostalog.
2. Njena uputstva najvišeg nivoa govore AccountManager-u da treba da obavesti vlasnika naloga o rezultatima zahteva za promenu naloga, koristeći funkciju notify_account_owner.
3. Sažeta definicija funkcije uključuje njene:

- ime
- opis
- parametre message: { type: ”string” }
- blok koji se izvršava kada se funkcija pozove

Nakon ažuriranja transkripta sa rezultatima funkcijskog bloka, ponovo se poziva metoda chat_completion. Ova metoda je odgovorna za slanje ažuriranog transkripta razgovora nazad AI modelu za dalju obradu. Ovaj proces nazivamo *konverzacionom petljom*.

Kada AI model primi novi zahtev za završetak četovanja sa ažuriranim transkriptom, ima pristup rezultatima prethodno izvršene funkcije. Može da analizira ove rezultate,

uključi ih u svoj proces odlučivanja i generiše sledeći odgovor ili akciju na osnovu kumulativnog konteksta razgovora. Može da izabere da izvrši dodatne funkcije na osnovu ažuriranog konteksta, ili može da generiše konačan odgovor na originalni upit ako utvrdi da nisu potrebni dodatni pozivi funkcija.

Opcionalni nastavak originalnog upita

Kada pošaljete rezultate alata nazad VJM-u i nastavite obradu originalnog upita, AI koristi te rezultate da ili pozove dodatne funkcije ili generiše konačan tekstualni odgovor.

Neki modeli kao što je Cohere-ov Command-R mogu da citiraju specifične alate koje su koristili u svojim odgovorima, pružajući dodatnu transparentnost i mogućnost praćenja.

U zavisnosti od modela koji se koristi, rezultati poziva funkcije će živeti u porukama transkripta koje imaju svoju posebnu ulogu ili će biti prikazani u nekoj drugoj sintaksi. Ali važan deo je da ti podaci budu u transkriptu, kako bi ih AI mogao razmotriti dok odlučuje šta dalje da radi.

Česta (i potencijalno skupa) greška je zaboraviti dodavanje rezultata funkcije u transkript pre nastavka četovanja. Kao rezultat toga, AI će biti upitan na suštinski isti način kao pre nego što je prvi put pozvao funkciju. Drugim rečima, što se AI-ja tiče, on još nije pozvao funkciju. Tako da je poziva ponovo. I ponovo. I ponovo, zauvek dok ga ne prekinete. Nadajmo se da vaš kontekst nije bio prevelik, a vaš model nije bio preskup!

Najbolje prakse za upotrebu alata

Da biste najbolje iskoristili upotrebu alata, razmotrite sledeće najbolje prakse.

Opisne definicije

Obezbedite jasna i opisna imena i opise za svaki alat i njegove ulazne parametre. Ovo pomaže VJM-u da bolje razume svrhu i mogućnosti svakog alata.

> Iz iskustva vam mogu reći da se uobičajena mudrost koja kaže da je "imenovanje teško" primenjuje i ovde; video sam dramatično različite rezultate od VJM-ova samo promenom imena funkcija ili formulacije opisa. Ponekad uklanjanje opisa *poboljšava* performanse.

Obrada rezultata alata

Kada prosleđujete rezultate alata nazad VJM-u, pobrinite se da su dobro strukturirani i sveobuhvatni. Koristite smislene ključeve i vrednosti za predstavljanje izlaza svakog alata. Eksperimentišite sa različitim formatima i vidite koji najbolje radi, od JSON-a do običnog teksta.

Interpreter rezultata se bavi ovim izazovom koristeći AI za analizu rezultata i pružanje objašnjenja, rezimea ili ključnih zaključaka prilagođenih ljudima.

Rukovanje greškama

Implementirajte robustne mehanizme za rukovanje greškama kako biste obradili slučajeve kada VJM može generisati nevažeće ili nepodržane ulazne parametre za pozive alata. Elegantno rukujte i oporavite se od bilo kakvih grešaka koje se mogu pojaviti tokom izvršavanja alata.

Jedna izuzetno lepa osobina AI-ja je da razume poruke o greškama! Što znači da ako radite u brzom i površnom mentalitetu, možete jednostavno uhvatiti bilo koje izuzetke generisane u implementaciji alata i proslediti ih nazad AI-ju tako da zna šta se dogodilo!

Na primer, evo pojednostavljene verzije implementacije Google pretrage u Olympia-i:

```
  def google_search(conversation, params)
    conversation.update_cstatus("Searching Google...")
    query = params[:query]
    search = GoogleSearch.new(query).get_hash

    conversation.update_cstatus("Summarizing results...")
    SummarizeKnowledgeGraph.new.perform(conversation, search.to_json)
  rescue StandardError => e
    Honeybadger.notify(e)
    { error: e.message }.inspect
  end
```

Google pretrage u Olympiji su dvostepeni proces. Prvo se izvrši pretraga, zatim se sumiraju rezultati. Ako dođe do greške, bez obzira kakva je, poruka o grešci se pakuje i šalje nazad veštačkoj inteligenciji. Ova tehnika je temelj praktično svih obrazaca *Inteligentnog rukovanja greškama.*

Na primer, recimo da GoogleSearch API poziv ne uspe zbog 503 Service Unavailable izuzetka. To se propagira do najvišeg nivoa obrade grešaka, a opis greške se šalje nazad veštačkoj inteligenciji kao rezultat funkcijskog poziva. Umesto da korisniku prikaže prazan ekran ili tehničku grešku, veštačka inteligencija kaže nešto poput "Žao mi je, ali trenutno ne mogu da pristupim svojim Google pretraživačkim mogućnostima. Mogu pokušati ponovo kasnije, ako želite."

Ovo može delovati kao pametan trik, ali razmotrite drugačiju vrstu greške, onu gde veštačka inteligencija poziva eksterni API i ima direktnu kontrolu nad parametrima koje prosleđuje API-ju. Možda je napravila grešku u načinu na koji je generisala te parametre? Pod uslovom da je poruka o grešci iz eksternog API-ja dovoljno detaljna, prosleđivanje poruke o grešci nazad veštačkoj inteligenciji koja je izvršila poziv znači da ona može preispitati te parametre i pokušati ponovo. Automatski. Bez obzira na to kakva je greška bila.

Sada razmislite šta bi bilo potrebno da se replicira takva vrsta robusnog rukovanja greškama u *normalnom* kodu. To je praktično nemoguće.

Iterativno poboljšanje

Ako VJM ne preporučuje odgovarajuće alate ili generiše suboptimalne odgovore, iterativno radite na definicijama alata, opisima i ulaznim parametrima. Kontinuirano usavršavajte i poboljšavajte postavke alata na osnovu uočenog ponašanja i željenih ishoda.

1. Počnite sa jednostavnim definicijama alata: Počnite definisanjem alata sa jasnim i konciznim imenima, opisima i ulaznim parametrima. U početku izbegavajte prekomplikovano podešavanje alata i fokusirajte se na osnovnu funkcionalnost. Na primer, ako želite da sačuvate rezultate analize sentimenta, počnite sa osnovnom definicijom poput:

```
{
  "name": "save_sentiment_score",
  "description": "Analyze user-provided text and generate sentiment score",
  "parameters": {
    "type": "object",
    "properties": {
      "score": {
        "type": "float",
        "description": "sentiment score from -1 (negative) to 1 (positive)"
      }
    },
    "required": ["score"]
  }
}
```

2. Testirajte i posmatrajte: Kada postavite početne definicije alata, testirajte ih različitim upitima i posmatrajte kako VJM komunicira sa alatom. Obratite pažnju na kvalitet i relevantnost generisanih odgovora. Ako VJM generiše suboptimalne odgovore, vreme je da se definicije alata usavrše.
3. Usavršite opise: Ako VJM pogrešno tumači svrhu alata, pokušajte da usavršite opis alata. Pružite više konteksta, primera ili pojašnjenja kako biste usmerili

VJM ka efikasnom korišćenju alata. Na primer, možete ažurirati opis alata za analizu sentimenta kako bi se preciznije odnosio na *emocionalni ton* teksta koji se analizira:

```
{
  "name": "save_sentiment_score",
  "description": "Determine the overall emotional tone of a piece of text,
   such as customer reviews, social media posts, or feedback comments.",
  ...
}
```

4. Prilagodite ulazne parametre: Ako LLM generiše nevažeće ili neodgovarajuće ulazne parametre za alat, razmotrite prilagođavanje definicija parametara. Dodajte konkretnije uslove, pravila validacije ili primere kako biste pojasnili očekivani format unosa.
5. Iteriranje na osnovu povratnih informacija: Kontinuirano pratite performanse vaših alata i prikupljajte povratne informacije od korisnika i zainteresovanih strana. Koristite ove povratne informacije da identifikujete oblasti za poboljšanje i napravite iterativna poboljšanja definicija alata. Na primer, ako korisnici prijave da analiza ne obrađuje sarkazam dobro, možete dodati napomenu u opis:

```
{
  "name": "save_sentiment_score",
  "description": "Analyze the sentiment of a given text and return a sentiment
   score between -1 (negative) and 1 (positive). Note: Sarcasm should be
   considered negative.",
  ...
}
```

Iterativnim poboljšavanjem definicija alata na osnovu uočenog ponašanja i povratnih informacija, možete postepeno unapređivati performanse i efikasnost vaše aplikacije zasnovane na veštačkoj inteligenciji. Zapamtite da definicije alata treba da budu jasne, sažete i fokusirane na specifičan zadatak. Redovno testirajte i proveravajte interakcije alata kako biste osigurali da su u skladu sa željenim ishodima.

Komponovanje i ulančavanje alata

Jedan od najmoćnijih aspekata upotrebe alata, koji je do sada samo nagovešten, jeste mogućnost komponovanja i ulančavanja više alata zajedno kako bi se izvršili složeni zadaci. Pažljivim dizajniranjem definicija alata i njihovih ulaznih/izlaznih formata, možete kreirati ponovno upotrebljive gradivne blokove koji se mogu kombinovati na različite načine.

Razmotrimo primer gde gradite protok analize podataka za vašu aplikaciju zasnovanu na veštačkoj inteligenciji. Mogli biste imati sledeće alate:

1. **DataRetrieval**: Alat koji preuzima podatke iz baze podataka ili API-ja na osnovu određenih kriterijuma.
2. **DataProcessing**: Alat koji vrši proračune, transformacije ili agregacije nad preuzetim podacima.
3. **DataVisualization**: Alat koji predstavlja obrađene podatke u formatu prilagođenom korisniku, kao što su grafikoni ili dijagrami.

Ulančavanjem ovih alata, možete kreirati moćan tok rada koji preuzima relevantne podatke, obrađuje ih i predstavlja rezultate na smislen način. Evo kako bi tok rada sa alatima mogao izgledati:

1. VJM prima upit korisnika koji traži uvid u podatke o prodaji za određenu kategoriju proizvoda.
2. VJM bira alat DataRetrieval i generiše odgovarajuće ulazne parametre za preuzimanje relevantnih podataka o prodaji iz baze podataka.
3. Preuzeti podaci se "prosleđuju" alatu DataProcessing, koji izračunava metrike kao što su ukupan prihod, prosečna prodajna cena i stopa rasta.
4. Obrađene podatke zatim koristi alat DataVisualization, koji kreira vizuelno privlačan grafikon ili dijagram za predstavljanje uvida, prosleđujući URL grafikona nazad VJM-u.

5. Na kraju, VJM generiše formatiran odgovor na upit korisnika koristeći markdown, uključujući vizualizovane podatke i rezime ključnih nalaza.

Komponovanjem ovih alata zajedno, možete kreirati besprekoran tok analize podataka koji se lako može integrisati u vašu aplikaciju. Lepota ovog pristupa je u tome što se svaki alat može razvijati i testirati nezavisno, a zatim kombinovati na različite načine za rešavanje raznih problema.

Da biste omogućili glatko komponovanje i ulančavanje alata, važno je definisati jasne ulazne i izlazne formate za svaki alat.

Na primer, alat DataRetrieval može prihvatiti parametre kao što su detalji veze sa bazom podataka, ime tabele i uslovi upita, i vratiti skup rezultata kao strukturirani JSON objekat. Alat DataProcessing zatim može očekivati ovaj JSON objekat kao ulaz i proizvesti transformisani JSON objekat kao izlaz. Standardizacijom protoka podataka između alata možete osigurati kompatibilnost i mogućnost ponovne upotrebe.

Dok dizajnirate svoj ekosistem alata, razmišljajte o tome kako se različiti alati mogu kombinovati za rešavanje uobičajenih slučajeva upotrebe u vašoj aplikaciji. Razmotrite kreiranje alata visokog nivoa koji obuhvataju uobičajene tokove rada ili poslovnu logiku, čineći lakšim za VJM da ih efikasno odabere i koristi.

Zapamtite, snaga upotrebe alata leži u fleksibilnosti i modularnosti koju pruža. Raščlanjivanjem složenih zadataka na manje, ponovno upotrebljive alate, možete kreirati robusnu i prilagodljivu aplikaciju zasnovanu na veštačkoj inteligenciji koja može da se nosi sa širokim spektrom izazova.

Budući pravci

Kako se oblast razvoja aplikacija zasnovanih na veštačkoj inteligenciji razvija, možemo očekivati dalja unapređenja u mogućnostima upotrebe alata. Neki potencijalni budući pravci uključuju:

1. **Višestruka upotreba alata**: VJM-ovi će možda moći da odluče koliko puta treba da koriste alate kako bi generisali zadovoljavajući odgovor. Ovo bi moglo uključivati više rundi odabira i izvršavanja alata na osnovu međurezultata.
2. **Predefinisani alati**: AI platforme mogu pružiti skup predefinisanih alata koje programeri mogu koristiti odmah, kao što su Python interpreteri, alati za pretragu veba ili uobičajene pomoćne funkcije.
3. **Besprekorna integracija**: Kako upotreba alata postaje sve zastupljenija, možemo očekivati bolju integraciju između AI platformi i popularnih razvojnih okvira, olakšavajući programerima da uključe upotrebu alata u svoje aplikacije.

Upotreba alata je moćna tehnika koja omogućava programerima da iskoriste pun potencijal VJM-ova u aplikacijama zasnovanim na veštačkoj inteligenciji. Povezivanjem VJM-ova sa spoljnim alatima i resursima, možete kreirati dinamičnije, inteligentnije i kontekstualno svesnije sisteme koji se mogu prilagoditi potrebama korisnika i pružiti vredne uvide i akcije.

Iako upotreba alata nudi ogromne mogućnosti, važno je biti svestan potencijalnih izazova i razmatranja. Jedan ključni aspekt je upravljanje složenošću interakcija alata i osiguravanje stabilnosti i pouzdanosti celokupnog sistema. Morate se nositi sa scenarijima gde pozivi alata mogu da ne uspeju, vrate neočekivane rezultate ili imaju implikacije na performanse. Dodatno, trebalo bi razmotriti mere bezbednosti i kontrole pristupa kako bi se sprečila neovlašćena ili zlonamerna upotreba alata. Pravilno rukovanje greškama, vođenje evidencije i mehanizmi praćenja su ključni za održavanje integriteta i performansi vaše aplikacije zasnovane na veštačkoj inteligenciji.

Dok istražujete mogućnosti upotrebe alata u svojim projektima, zapamtite da treba početi sa jasnim ciljevima, dizajnirati dobro strukturirane definicije alata i iterirati na osnovu povratnih informacija i rezultata. Uz pravi pristup i način razmišljanja, upotreba alata može otključati nove nivoe inovacije i vrednosti u vašim aplikacijama vođenim veštačkom inteligencijom

Obrada toka podataka

Streaming podataka preko HTTP-a, takođe poznat kao server-sent events (SSE), je mehanizam gde server kontinuirano šalje podatke klijentu kako oni postaju dostupni, bez potrebe da ih klijent eksplicitno zatraži. Pošto se AI odgovor generiše postepeno, ima smisla obezbediti responzivno korisničko iskustvo prikazivanjem AI izlaza tokom njegovog generisanja. I zapravo, svi API-ji AI provajdera koje poznajem nude streaming odgovore kao opciju u svojim krajnjim tačkama za kompletiranje.

Razlog zašto se ovo poglavlje pojavljuje ovde u knjizi, odmah nakon Korišćenje alata, je zbog toga koliko moćno može biti kombinovanje korišćenja alata sa živim AI odgovorima korisnicima. To omogućava dinamička i interaktivna iskustva gde AI može da obradi korisnički unos, koristi različite alate i funkcije po svom nahođenju, i zatim pruža odgovore u realnom vremenu.

Da bi se postigla ova besprekorna interakcija, potrebno je napisati rukovaoce toka koji mogu da otpremaju pozive alatnih funkcija koje je pozvao AI, kao i običan tekstualni

izlaz krajnjem korisniku. Potreba za petljom nakon obrade alatne funkcije dodaje zanimljiv izazov ovom zadatku.

Implementacija ReplyStream-a

Da bismo pokazali kako se može implementirati obrada toka, ovo poglavlje će detaljno obraditi pojednostavljenu verziju klase ReplyStream koja se koristi u Olympia-i. Instance ove klase mogu se proslediti kao stream parametar u AI klijentskim bibliotekama kao što su ruby-openai i openrouter

Evo kako koristim ReplyStream u Olympia-inom PromptSubscriber-u, koji osluškuje preko Wisper-a kreiranje novih korisničkih poruka.

```
1  class PromptSubscriber
2    include Raix::ChatCompletion
3    include Raix::PromptDeclarations
4
5    # many other declarations omitted...
6
7    prompt text: -> { user_message.content },
8           stream: -> { ReplyStream.new(self) },
9           until: -> { bot_message.complete? }
10
11   def message_created(message) # invoked by Wisper
12     return unless message.role.user? && message.content?
13
14     # rest of the implementation omitted...
```

Pored context reference na pretplatnika prompta koji ga je instancirao, klasa ReplyStream takođe ima promenljive instance za čuvanje bafera primljenih podataka, kao i nizove za praćenje imena funkcija i argumenata koji se pozivaju tokom obrade toka.

```
class ReplyStream
  attr_accessor :buffer, :f_name, :f_arguments, :context

  delegate :bot_message, :dispatch, to: :context

  def initialize(context)
    self.context = context
    self.buffer = []
    self.f_name = []
    self.f_arguments = []
  end

  def call(chunk, bytesize = nil)
    # ...
  end

  # ...
end
```

Metoda initialize postavlja početno stanje instance ReplyStream, inicijalizujući bafer, kontekst i druge promenljive.

Metoda call je glavna ulazna tačka za obradu protočnih podataka. Ona prima chunk podataka (predstavljen kao heš) i opcionalni parametar bytesize, koji u našem primeru nije iskorišćen. Unutar ove metode, klasa koristi uparivanje obrazaca za rukovanje različitim scenarijima na osnovu strukture primljenog dela podataka.

Pozivanje deep_symbolize_keys na delu podataka pomaže da uparivanje obrazaca bude elegantnije, omogućavajući nam da radimo sa simbolima umesto sa stringovima.

```
def call(chunk, _bytesize)
  case chunk.deep_symbolize_keys

  in { # match function name
    choices: [
      {
        delta: {
          tool_calls: [
            { index: index, function: {name: name} }
          ]
        }
      }
    ] }

    f_name[index] = name
```

Prvi obrazac koji tražimo je poziv alata zajedno sa njegovim pripadajućim imenom funkcije. Ako ga otkrijemo, smeštamo ga u niz f_name. Imena funkcija čuvamo u indeksiranom nizu, jer model može da vrši paralelno pozivanje funkcija, šaljući više od jedne funkcije na izvršavanje istovremeno.

> Paralelno pozivanje funkcija je sposobnost AI modela da izvršava više poziva funkcija zajedno, omogućavajući da se efekti i rezultati ovih poziva funkcija rešavaju paralelno. Ovo je posebno korisno ako funkcije dugo traju, i smanjuje broj razmena sa API-jem, što zauzvrat može uštedeti značajnu količinu potrošnje tokena.

Zatim moramo pronaći podudaranje za argumente koji odgovaraju pozivima funkcija.

```
  in { # match arguments
    choices: [
      {
        delta: {
          tool_calls: [
            {
              index: index, function: {arguments: argument }
            }
          ]
        }
      }
    ]}

    f_arguments[index] ||= "" # initialize if not already
    f_arguments[index] << argument
```

Slično kao što smo postupili sa imenima funkcija, argumente smeštamo u indeksirani niz.

Sledeće, pratimo normalne poruke namenjene korisniku, koje će stizati sa servera token po token i biti dodeljene promenljivoj new_content. Takođe moramo da pratimo finish_reason. On će biti nil sve do poslednjeg dela izlazne sekvence.

```
  in {
    choices: [
      { delta: {content: new_content}, finish_reason: finish_reason }
    ]}

    # you could transmit every chunk to the user here...
    buffer << new_content.to_s

    if finish_reason.present?
      finalize
    elsif new_content.to_s.match?(/\n\n/)
      send_to_client # ...or buffer and transmit once per paragraph
    end
```

Važno je da dodamo izraz za podudaranje obrazaca kako bismo obradili poruke o greškama koje šalje provajder AI modela. U lokalnim razvojnim okruženjima, podižemo izuzetak, ali u produkciji, beležimo grešku i završavamo.

```
  in { error: { message: } }
    if Rails.env.local?
      raise message
    else
      Honeybadger.notify("AI Error: #{message}")
      finalize
    end
```

Završna else grana case naredbe će se izvršiti ako se nijedan od prethodnih obrazaca nije poklopio. To je samo mera predostrožnosti kako bismo otkrili ako AI model počne da nam šalje neprepoznate delove.

```
  else
    Honeybadger.notify("Unrecognized Chunk: #{chunk}")
  end
end
```

Metoda send_to_client je zadužena za slanje baferovanog sadržaja klijentu. Ona proverava da bafer nije prazan, ažurira sadržaj poruke bota, renderuje poruku bota i čuva sadržaj u bazi podataka kako bi se osigurala postojanost podataka.

```
def send_to_client
  # no need to process pure whitespace
  return if buffer.join.squish.blank?

  # set the buffer content on the bot message
  content = buffer.join
  bot_message.content = content

  # save to database so that we never lose data
  # even if the stream doesn't terminate correctly
  bot_message.update_column(:content, content)

  # update content via websocket
  ConversationRenderer.update(bot_message)
end
```

finalize metoda se poziva kada je obrada toka završena. Ona otprema pozive funkcija ako su neki primljeni tokom toka, ažurira poruku bota sa konačnim sadržajem i drugim relevantnim informacijama, i resetuje istoriju poziva funkcija

```
def finalize
  if f_name.any?
    f_name.each_with_index do |name, index|
      # takes care of calling the function wherever it's implemented
      dispatch(name:, arguments: JSON.parse(f_arguments[index]))
    end

    # reset the function call history
    f_name.clear
    f_arguments.clear
  else
    content = buffer.join.presence
    bot_message.update!(content:, complete: true)
    ConversationRenderer.update(bot_message)
  end
end
```

Ako model odluči da pozove funkciju, potrebno je da "prosledite" taj poziv funkcije (naziv i argumente) na takav način da se izvrši i da se poruke function_call i function_result dodaju u transkript razgovora

Prema mom iskustvu, bolje je upravljati kreiranjem poruka funkcija na jednom mestu u vašoj bazi koda, umesto da se oslanjate na implementacije alata. To nije samo čistije rešenje, već ima i vrlo važan praktičan razlog: ako AI model pozove funkciju, a ne vidi rezultujuće poruke poziva i rezultata u transkriptu kada se petlja ponovi, *pozivaće istu funkciju iznova*. Potencijalno beskonačno. Zapamtite da je AI potpuno bez stanja, tako da ako ne prikažete te pozive funkcija nazad modelu, oni se nisu ni dogodili.

```ruby
# PromptSubscriber#dispatch

def dispatch(name:, arguments:)
  # adds a function_call message to the conversation transcript
  # plus dispatches to tool and returns result
  conversation.function_call!(name, arguments).then do |result|
    # add function result message to the transcript
    conversation.function_result!(name, result)
  end
end
```

Čišćenje istorije poziva funkcija nakon izvršavanja je podjednako važno kao i osiguravanje da se poziv i rezultati nađu u vašem transkriptu, kako ne biste stalno iznova pozivali iste funkcije svaki put kada se petlja izvrši.

"Konverzacijska petlja"

U klasi PromptSubscriber, koristimo metodu prompt iz modula PromptDeclarations da definišemo ponašanje konverzacijske petlje. Parametar until je postavljen na -> { bot_message.complete? }, što znači da će se petlja nastaviti sve dok bot_message ne bude označen kao završen.

```ruby
prompt text: -> { user_message.content },
       stream: -> { ReplyStream.new(self) },
       until: -> { bot_message.complete? }
```

Ali kada se bot_message označava kao završen? Ako ste zaboravili, pogledajte ponovo red 13 metode finalize.

Hajde da pregledamo celokupnu logiku obrade toka.

1. PromptSubscriber prima novu korisničku poruku preko metode message_created, koju poziva Wisper sistem za objavljivanje/pretplatu svaki put kada krajnji korisnik kreira novi prompt.
2. Klasna metoda prompt deklarativno definiše ponašanje logike četbot završetka za PromptSubscriber. AI model će izvršiti četbot završetak sa sadržajem korisničke poruke, novom instancom ReplyStream kao parametrom toka i određenim uslovom petlje.
3. AI model obrađuje prompt i počinje da generiše odgovor. Kako se odgovor prenosi u toku, metoda call instance ReplyStream se poziva za svaki deo podataka.
4. Ako AI model odluči da pozove pomoćnu funkciju, naziv funkcije i argumenti se izdvajaju iz dela i čuvaju u nizovima f_name i f_arguments.
5. Ako AI model generiše sadržaj vidljiv korisniku, on se skladišti u baferu i šalje klijentu preko metode send_to_client.
6. Kada se obrada toka završi, poziva se metoda finalize. Ako su tokom toka pozvane bilo koje pomoćne funkcije, one se otpremaju koristeći metodu dispatch klase PromptSubscriber.
7. Metoda dispatch dodaje poruku function_call u transkript razgovora, izvršava odgovarajuću pomoćnu funkciju i dodaje poruku function_result u transkript sa rezultatom poziva funkcije.
8. Nakon otpremanja pomoćnih funkcija, istorija poziva funkcija se briše kako bi se sprečili dupli pozivi funkcija u narednim petljama.
9. Ako nije pozvana nijedna pomoćna funkcija, metoda finalize ažurira bot_-message sa konačnim sadržajem, označava ga kao završen i šalje ažuriranu poruku klijentu.
10. Uslov petlje -> { bot_message.complete? } se procenjuje. Ako bot_message nije označen kao završen, petlja se nastavlja, i originalni prompt se ponovo šalje sa ažuriranim transkriptom razgovora.
11. Koraci 3-10 se ponavljaju dok se bot_message ne označi kao završen, što ukazuje da je AI model završio generisanje svog odgovora i da nema potrebe za

izvršavanjem dodatnih pomoćnih funkcija.

Implementacijom ove petlje razgovora, omogućavate AI modelu da se upusti u dvosmerni dijalog sa aplikacijom, izvršavajući pomoćne funkcije po potrebi i generišući odgovore vidljive korisniku dok razgovor ne dođe do prirodnog zaključka.

Kombinacija obrade toka i petlje razgovora omogućava dinamična i interaktivna AI iskustva, gde AI model može da obrađuje korisnički unos, koristi različite alate i funkcije, i pruža odgovore u realnom vremenu na osnovu konteksta razgovora koji se razvija.

Automatski Nastavak

Važno je biti svestan ograničenja AI izlaza. Većina modela ima maksimalni broj tokena koje mogu generisati u jednom odgovoru, što je određeno parametrom max_tokens. Ako AI model dostigne ovo ograničenje tokom generisanja odgovora, naglo će se zaustaviti i ukazati da je izlaz skraćen.

U toku odgovora sa AI platforme API-ja, možete otkriti ovu situaciju pregledanjem promenljive finish_reason u delu. Ako je finish_reason postavljen na ”length” (ili neku drugu ključnu vrednost specifičnu za model), to znači da je model dostigao svoje maksimalno ograničenje tokena tokom generisanja i da je izlaz skraćen.

Jedan od načina da se ova situacija elegantno reši i pruži besprekorno korisničko iskustvo je implementacija mehanizma automatskog nastavka u logici obrade toka. Dodavanjem obrasca za prepoznavanje razloga završetka vezanih za dužinu, možete odabrati da se petlja nastavi i automatski nastavi izlaz od mesta gde je stao.

Evo namerno pojednostavljenog primera kako možete modifikovati metodu call u klasi ReplyStream da podržava automatski nastavak:

```
LENGTH_STOPS = %w[length MAX_TOKENS]

def call(chunk, _bytesize)
  case chunk.deep_symbolize_keys
    # ...

    in {
      choices: [
        { delta: {content: new_content},
          finish_reason: finish_reason } ] }

      buffer << new_content.to_s

      if finish_reason.blank?
        send_to_client if new_content.to_s.match?(/\n\n/)
      elsif LENGTH_STOPS.include?(finish_reason)
        continue_cutoff
      else
        finalize
      end

    # ...
  end
end

private

def continue_cutoff
  conversation.bot_message!(buffer.join, visible: false)
  conversation.user_message!("please continue", visible: false)
  bot_message.update_column(:created_at, Time.current)
end
```

U ovoj modifikovanoj verziji, kada finish_reason ukazuje na odsečeni izlaz, umesto finalizacije toka, dodajemo par poruka u transkript bez finalizacije, pomeramo originalnu poruku vidljivu korisniku na "dno" transkripta ažuriranjem njenog created_at atributa, a zatim pustimo da se petlja nastavi, tako da AI nastavlja generisanje tamo gde je stao.

Zapamtite da je krajnja tačka za AI dopunu bez stanja. Ona "zna" samo ono što joj kažete

preko transkripta. U ovom slučaju, način na koji AI-u saopštavamo da je došlo do prekida je dodavanjem "nevidljivih" (za krajnjeg korisnika) poruka u transkript. Ipak, imajte na umu da je ovo namerno pojednostavljen primer. Prava implementacija bi morala da vrši dodatno upravljanje transkriptom kako bi se osiguralo da ne trošimo tokene i/ili zbunimo AI dupliranim porukama asistenta u transkriptu.

Prava implementacija automatskog nastavljanja bi takođe trebalo da ima takozvanu "logiku prekidača" kako bi se sprečilo nekontrolisano ponavljanje petlje. Razlog je taj što bi, uz određene vrste korisničkih upita i niske postavke max_tokens, AI mogao beskonačno da nastavlja da generiše izlaz vidljiv korisniku.

> Imajte na umu da svaka petlja zahteva poseban zahtev, i da svaki zahtev ponovo koristi ceo vaš transkript. Definitivno bi trebalo da razmotrite kompromise između korisničkog iskustva i korišćenja API-ja kada odlučujete da li da implementirate automatsko nastavljanje u vašoj aplikaciji. Automatsko nastavljanje može biti posebno opasno skupo, naročito kada se koriste premium komercijalni modeli.

Zaključak

Obrada toka podataka je kritičan aspekt izgradnje aplikacija pokretanih veštačkom inteligencijom koje kombinuju upotrebu alata sa živim AI odgovorima. Efikasnim upravljanjem tokovima podataka iz API-ja AI platformi, možete obezbediti neometano i interaktivno korisničko iskustvo, upravljati velikim odgovorima, optimizovati korišćenje resursa i elegantno upravljati greškama.

Prikazana klasa Conversation::ReplyStream demonstrira kako obrada toka može biti implementirana u Ruby aplikaciji koristeći uparivanje obrazaca i arhitekturu vođenu događajima. Razumevanjem i korišćenjem tehnika obrade toka, možete otključati pun

potencijal AI integracije u vašim aplikacijama i isporučiti moćna i privlačna korisnička iskustva.

Samozalečujući podaci

Samozalečujući podaci predstavljaju moćan pristup osiguravanju integriteta, konzistentnosti i kvaliteta podataka u aplikacijama korišćenjem mogućnosti velikih jezičkih modela (VJM). Ova kategorija obrazaca fokusira se na ideju korišćenja veštačke inteligencije za automatsko otkrivanje, dijagnostikovanje i ispravljanje anomalija, nekonzistentnosti ili grešaka u podacima, čime se smanjuje opterećenje programera i održava visok nivo pouzdanosti podataka.

U svojoj suštini, obrasci samozalečujućih podataka prepoznaju da su podaci životna snaga svake aplikacije, a osiguravanje njihove tačnosti i integriteta je ključno za pravilno funkcionisanje i korisničko iskustvo aplikacije. Međutim, upravljanje i održavanje kvaliteta podataka može biti složen i vremenski zahtevan zadatak, posebno kako aplikacije rastu u veličini i složenosti. Tu na scenu stupa moć veštačke inteligencije.

U obrascima samozalečujućih podataka, AI radnici se koriste za kontinuirano praćenje i analizu podataka vaše aplikacije. Ovi modeli imaju sposobnost da razumeju i

tumače obrasce, veze i anomalije unutar podataka. Koristeći svoje mogućnosti obrade i razumevanja prirodnog jezika, mogu identifikovati potencijalne probleme ili nekonzistentnosti u podacima i preduzeti odgovarajuće mere za njihovo ispravljanje.

Proces samozalečenja podataka obično uključuje nekoliko ključnih koraka:

1. **Praćenje podataka**: AI radnici konstantno prate tokove podataka, baze podataka ili sisteme za skladištenje aplikacije, tražeći bilo kakve znakove anomalija, nekonzistentnosti ili grešaka. Alternativno, možete aktivirati AI komponentu kao reakciju na izuzetak.
2. **Detekcija anomalija**: Kada se otkrije problem, AI radnik detaljno analizira podatke kako bi identifikovao specifičnu prirodu i obim problema. To može uključivati otkrivanje nedostajućih vrednosti, nekonzistentnih formata ili podataka koji krše predefinisana pravila ili ograničenja.
3. **Dijagnoza i korekcija**: Nakon što se problem identifikuje, AI radnik koristi svoje znanje i razumevanje domena podataka da odredi odgovarajući tok akcije. To može uključivati automatsko ispravljanje podataka, popunjavanje nedostajućih vrednosti ili označavanje problema za ljudsku intervenciju ako je potrebno.
4. **Kontinuirano učenje (opciono, zavisno od slučaja upotrebe)**: Kako vaš AI radnik nailazi na različite probleme sa podacima i rešava ih, može generisati izlaz koji opisuje šta se dogodilo i kako je reagovao. Ovi metapodaci se mogu uneti u procese učenja koji vam (i možda osnovnom modelu, putem finog podešavanja) omogućavaju da vremenom postanete efikasniji u identifikovanju i rešavanju anomalija podataka.

Automatskim otkrivanjem i ispravljanjem problema sa podacima, možete osigurati da vaša aplikacija radi sa visokokvalitetnim, pouzdanim podacima. Ovo smanjuje rizik od grešaka, nekonzistentnosti ili bagova vezanih za podatke koji mogu uticati na funkcionalnost aplikacije ili korisničko iskustvo.

Kada imate AI radnike koji se bave zadatkom praćenja i korekcije podataka, možete usmeriti svoje napore na druge kritične aspekte aplikacije. Ovo štedi vreme i resurse

koji bi inače bili potrošeni na ručno čišćenje i održavanje podataka. Zapravo, kako vaše aplikacije rastu u veličini i složenosti, ručno upravljanje kvalitetom podataka postaje sve izazovnije. Obrasci "Samozalečujućih podataka" efikasno skaliraju koristeći moć veštačke inteligencije za obradu velikih količina podataka i otkrivanje problema u realnom vremenu.

Zbog svoje prirode, AI modeli se mogu prilagoditi promenljivim obrascima podataka, šemama ili zahtevima tokom vremena uz malo ili nimalo nadzora. Sve dok njihove direktive pružaju adekvatno usmeravanje, posebno u pogledu željenih rezultata, vaša aplikacija može da evoluira i upravlja novim scenarijima podataka bez potrebe za obimnom ručnom intervencijom ili izmenama koda.

Obrasci samozalečujućih podataka se dobro uklapaju sa drugim kategorijama obrazaca o kojima smo razgovarali, kao što je "Mnoštvo radnika". Mogućnost samozalečenja podataka može se posmatrati kao specijalizovana vrsta radnika koji se fokusira specifično na osiguravanje kvaliteta i integriteta podataka. Ova vrsta radnika radi uporedo sa drugim AI radnicima, pri čemu svaki doprinosi različitim aspektima funkcionalnosti aplikacije.

Implementacija obrazaca samozalečujućih podataka u praksi zahteva pažljivo projektovanje i integraciju AI modela u arhitekturu aplikacije. Zbog rizika od gubitka i oštećenja podataka, trebalo bi da definišete jasne smernice za način na koji ćete koristiti ovu tehniku. Takođe bi trebalo da razmotrite faktore kao što su performanse, skalabilnost i bezbednost podataka.

Praktična studija slučaja: Popravljanje neispravnog JSON-a

Jedan od najpraktičnijih i najjednostavnijih načina za korišćenje samozalečujućih podataka je takođe vrlo jednostavan za objašnjenje: popravljanje neispravnog JSON-a.

Ova tehnika se može primeniti na uobičajeni izazov rešavanja nesavršenih ili nekonzistentnih podataka koje generišu VJM-ovi, kao što je neispravan JSON, i pruža pristup za automatsko otkrivanje i ispravljanje ovih problema.

U Olympiji redovno se susrećem sa scenarijima gde LLM-ovi generišu JSON podatke koji nisu potpuno ispravni. Ovo se može dogoditi iz različitih razloga, kao što je dodavanje komentara pre ili posle stvarnog JSON koda od strane LLM-a, ili uvođenje sintaksnih grešaka poput nedostajućih zareza ili neukinutih dvostrukih navodnika. Ovi problemi mogu dovesti do grešaka parsiranja i prouzrokovati prekide u funkcionalnosti aplikacije.

Da bih rešio ovaj problem, implementirao sam praktično rešenje u obliku JsonFixer klase. Ova klasa otelotvoruje obrazac "Samozalečujućih podataka" tako što uzima neispravan JSON kao ulaz i koristi LLM da ga popravi, istovremeno čuvajući što je moguće više informacija i prvobitne namere.

```
class JsonFixer
  include Raix::ChatCompletion

  def call(bad_json, error_message)
    raise "No data provided" if bad_json.blank? || error_message.blank?

    transcript << {
        system: "Consider user-provided JSON that generated a parse
                 exception. Do your best to fix it while preserving the
                 original content and intent as much as possible." }
    transcript << { user: bad_json }
    transcript << { assistant: "What is the error message?"}
    transcript << { user: error_message }
    transcript << { assistant: "Here is the corrected JSON\n```json\n" }

    self.stop = ["```"]

    chat_completion(json: true)
  end

  def model
    "mistralai/mixtral-8x7b-instruct:nitro"
  end
```

```
24 end
```

Obratite pažnju kako JsonFixer koristi Ventriloquist za usmeravanje AI odgovora.

Proces samoopravljanja JSON podataka funkcioniše na sledeći način:

1. **Generisanje JSON-a**: LLM se koristi za generisanje JSON podataka na osnovu određenih upita ili zahteva. Međutim, zbog prirode LLM-ova, generisani JSON ne mora uvek biti potpuno validan. JSON parser će naravno izbaciti ParserError ako mu date nevažeći JSON.

```
1 begin
2   JSON.parse(llm_generated_json)
3 rescue JSON::ParserError => e
4   JsonFixer.new.call(llm_generated_json, e.message)
5 end
```

Imajte na umu da se poruka o grešci takođe prosleđuje pozivu JSONFixer-a tako da ne mora u potpunosti da pretpostavlja šta nije u redu sa podacima, posebno zato što će parser često tačno reći šta nije u redu.

2. **Korekcija zasnovana na VJM-u**: Klasa JSONFixer šalje neispravan JSON nazad VJM-u (LLM), zajedno sa specifičnim upitom ili instrukcijom da popravi JSON uz maksimalno očuvanje originalnih informacija i namere. VJM, obučen na ogromnim količinama podataka i sa razumevanjem JSON sintakse, pokušava da ispravi greške i generiše validan JSON string. Ograđivanje odgovora se koristi za ograničavanje izlaza VJM-a, a mi biramo Mixtral 8x7B kao AI model, jer je posebno dobar za ovakvu vrstu zadatka.

3. **Validacija i integracija**: Popravljeni JSON string koji vraća VJM parsira sama klasa JSONFixer, jer je pozvala chat_completion(json: true). Ako popravljeni JSON prođe validaciju, integriše se nazad u tok rada aplikacije, omogućavajući aplikaciji da nastavi sa obradom podataka bez prekida. Loš JSON je "izlečen".

Iako sam napisao i prepisao sopstvenu JSONFixer implementaciju nekoliko puta, sumnjam da je ukupno vreme uloženo u sve te verzije više od sat ili dva.

Imajte na umu da je očuvanje namere ključni element svakog obrasca samozalečujućih podataka. Proces korekcije zasnovan na VJM-u ima za cilj da očuva originalne informacije i nameru generisanog JSON-a što je više moguće. Ovo osigurava da popravljeni JSON zadržava svoje semantičko značenje i može se efikasno koristiti u kontekstu aplikacije.

Ova praktična implementacija pristupa "Samozalečujućih podataka" u Olympiji jasno pokazuje kako se AI, posebno VJM-ovi, mogu iskoristiti za rešavanje stvarnih izazova sa podacima. Ona pokazuje snagu kombinovanja tradicionalnih programerskih tehnika sa AI mogućnostima za izgradnju robusnih i efikasnih aplikacija.

Postelov zakon i obrazac "Samozalečujućih podataka"

"Samozalečujući podaci", kao što je prikazano kroz klasu JSONFixer, dobro se uklapa sa principom poznatim kao Postelov zakon, koji se takođe naziva Princip robusnosti. Postelov zakon glasi:

"Budite konzervativni u onome što radite, budite liberalni u onome što prihvatate od drugih."

Ovaj princip, koji je prvobitno artikulisao Jon Postel, pionir ranog Interneta, naglašava važnost izgradnje sistema koji su tolerantni prema raznovrsnim ili čak blago netačnim ulazima, dok istovremeno održavaju strogo pridržavanje određenih

protokola pri slanju izlaza.

U kontekstu "Samozalečujućih podataka", klasa JSONFixer otelotvoruje Postelov zakon time što je liberalna u prihvatanju pokvarenih ili nesavršenih JSON podataka generisanih od strane VJM-ova. Ne odbacuje odmah i ne otkazuje kada naiđe na JSON koji se strogo ne pridržava očekivanog formata. Umesto toga, zauzima tolerantan pristup i pokušava da popravi JSON koristeći snagu VJM-ova.

Budući liberalna u prihvatanju nesavršenog JSON-a, klasa JSONFixer pokazuje robusnost i fleksibilnost. Ona priznaje da podaci u stvarnom svetu često dolaze u različitim oblicima i možda se neće uvek pridržavati strogih specifikacija. Gracioznim rukovanjem i ispravljanjem ovih odstupanja, klasa osigurava da aplikacija može nastaviti da funkcioniše glatko, čak i u prisustvu nesavršenih podataka.

S druge strane, klasa JSONFixer se takođe pridržava konzervativnog aspekta Postelovog zakona kada je reč o izlazu. Nakon popravljanja JSON-a pomoću VJM-ova, klasa validira ispravljeni JSON kako bi osigurala da se strogo pridržava očekivanog formata. Ona održava integritet i tačnost podataka pre nego što ih prosledi drugim delovima aplikacije. Ovaj konzervativni pristup garantuje da je izlaz klase JSONFixer pouzdan i konzistentan, promovišući interoperabilnost i sprečavajući širenje grešaka.

Zanimljivosti o Jonu Postelu:

- Jon Postel (1943-1998) bio je američki računarski naučnik koji je igrao ključnu ulogu u razvoju Interneta. Bio je poznat kao "Bog Interneta" zbog svojih značajnih doprinosa osnovnim protokolima i standardima.
- Postel je bio urednik serije dokumenata Request for Comments (RFC), koja predstavlja niz tehničkih i organizacionih napomena o Internetu. Autor je ili koautor preko 200 RFC-ova, uključujući fundamentalne protokole kao što su TCP, IP i SMTP.
- Pored svojih tehničkih doprinosa, Postel je bio poznat po svom skromnom i kolaborativnom pristupu. Verovao je u važnost postizanja konsenzusa i

zajedničkog rada na izgradnji robusne i interoperabilne mreže.
- Postel je služio kao direktor Odeljenja za računarske mreže na Institutu za informacione nauke (ISI) Univerziteta Južne Kalifornije (USC) od 1977. do svoje prerane smrti 1998. godine.
- U znak priznanja za njegove ogromne doprinose, Postelu je posthumno dodeljena prestižna Tjuringova nagrada 1998. godine, koja se često naziva "Nobelova nagrada za računarstvo."

Klasa JSONFixer promoviše robusnost, fleksibilnost i interoperabilnost, što su bile osnovne vrednosti koje je Postel podržavao tokom svoje karijere. Izgradnjom sistema koji su tolerantni na nesavršenosti, uz istovremeno strogo pridržavanje protokola, možemo kreirati aplikacije koje su otpornije i prilagodljivije u suočavanju sa izazovima stvarnog sveta.

Razmatranja i kontraindikacije

Primenljivost pristupa samooporavljajućih podataka u potpunosti zavisi od vrste podataka kojima vaša aplikacija upravlja. Postoji razlog zašto možda ne želite da jednostavno prepravite JSON.parse da automatski ispravlja *sve JSON greške parsiranja* u vašoj aplikaciji: nisu sve greške takve da se mogu ili trebaju automatski ispraviti.

Samooporavljanje je posebno problematično kada je povezano sa regulatornim zahtevima ili zahtevima usklađenosti koji se odnose na rukovanje i obradu podataka. Neke industrije, poput zdravstva i finansija, imaju tako stroge propise u vezi sa integritetom podataka i mogućnošću revizije da bilo kakva "crna kutija" korekcija podataka bez odgovarajućeg nadzora ili evidencije može prekršiti ove propise. Ključno je osigurati da se sve tehnike samooporavljanja podataka koje osmislite usklade sa važećim pravnim i regulatornim okvirima.

Primena tehnika samooporavljanja podataka, posebno onih koje uključuju AI modele,

takođe može imati veliki uticaj na performanse aplikacije i korišćenje resursa. Obrada velikih količina podataka kroz AI modele za detekciju i korekciju grešaka može biti računarski zahtevna. Važno je proceniti kompromise između prednosti samooporavljajućih podataka i povezanih troškova performansi i resursa.

To rečeno, hajde da zaronimo u faktore koji su uključeni u odlučivanje kada i gde primeniti ovaj moćan pristup.

Kritičnost podataka

Kada razmatramo primenu tehnika samooporavljanja podataka, ključno je proceniti kritičnost podataka koji se obrađuju. Nivo kritičnosti se odnosi na važnost i osetljivost podataka u kontekstu vaše aplikacije i njenog poslovnog domena.

U nekim slučajevima, automatsko ispravljanje grešaka u podacima možda nije prikladno, posebno ako su podaci veoma osetljivi ili imaju pravne implikacije. Na primer, razmotrite sledeće scenarije:

1. **Finansijske transakcije**: U finansijskim aplikacijama, kao što su bankarski sistemi ili platforme za trgovanje, tačnost podataka je od najveće važnosti. Čak i male greške u finansijskim podacima mogu imati značajne posledice, kao što su netačna stanja računa, pogrešno usmerena sredstva ili pogrešne odluke o trgovanju. U ovim slučajevima, automatske korekcije bez temeljne verifikacije i revizije mogu uvesti neprihvatljive rizike.
2. **Medicinski kartoni**: Zdravstvene aplikacije se bave veoma osetljivim i poverljivim podacima pacijenata. Netačnosti u medicinskim kartonima mogu imati ozbiljne implikacije za bezbednost pacijenata i odluke o lečenju. Automatsko modifikovanje medicinskih podataka bez odgovarajućeg nadzora i validacije od strane kvalifikovanih zdravstvenih radnika može prekršiti regulatorne zahteve i ugroziti dobrobit pacijenata.
3. **Pravni dokumenti**: Aplikacije koje rukuju pravnim dokumentima, kao što su ugovori, sporazumi ili sudski podnesci, zahtevaju strogu tačnost i integritet.

> Čak i male greške u pravnim podacima mogu imati značajne pravne posledice. Automatske korekcije u ovom domenu možda nisu prikladne, jer podaci često zahtevaju ručni pregled i verifikaciju od strane pravnih stručnjaka kako bi se osigurala njihova validnost i izvršnost.

U ovim kritičnim scenarijima podataka, rizici povezani sa automatskim korekcijama često prevazilaze potencijalne koristi. Posledice uvođenja grešaka ili netačnog modifikovanja podataka mogu biti ozbiljne, što dovodi do finansijskih gubitaka, pravnih odgovornosti, pa čak i štete po pojedince.

Kada se radi sa visoko kritičnim podacima, neophodno je dati prioritet procesima ručne verifikacije i validacije. Ljudski nadzor i stručnost su ključni u osiguravanju tačnosti i integriteta podataka. Automatizovane tehnike samooporavljanja se i dalje mogu koristiti za označavanje potencijalnih grešaka ili nedoslednosti, ali konačna odluka o korekcijama treba da uključuje ljudsku procenu i odobrenje.

Međutim, važno je napomenuti da nemaju svi podaci u aplikaciji isti nivo kritičnosti. U istoj aplikaciji mogu postojati podskupovi podataka koji su manje osetljivi ili imaju manji uticaj ako dođe do grešaka. U takvim slučajevima, tehnike samooporavljanja podataka mogu se selektivno primeniti na te specifične podskupove podataka, dok kritični podaci ostaju podložni ručnoj verifikaciji.

Ključ je pažljivo proceniti kritičnost svake kategorije podataka u vašoj aplikaciji i definisati jasne smernice i procese za rukovanje korekcijama na osnovu povezanih rizika i implikacija. Razlikovanjem između kritičnih (npr. knjigovodstveni podaci, medicinski kartoni) i nekritičnih podataka (npr. poštanske adrese, upozorenja o resursima), možete postići ravnotežu između korišćenja prednosti tehnika samooporavljanja podataka gde je to prikladno i održavanja stroge kontrole i nadzora gde je to neophodno.

Na kraju, odluka o primeni tehnika samooporavljanja podataka na kritične podatke treba da se donese u konsultaciji sa stručnjacima iz date oblasti, pravnim savetnicima i drugim relevantnim zainteresovanim stranama. Neophodno je uzeti u obzir specifične zahteve,

propise i rizike povezane sa podacima vaše aplikacije i uskladiti strategije korekcije podataka u skladu sa tim.

Ozbiljnost grešaka

Prilikom primene tehnika samooporavljanja podataka, važno je proceniti ozbiljnost i uticaj grešaka u podacima. Nisu sve greške jednake, i odgovarajući tok akcije može varirati u zavisnosti od ozbiljnosti problema.

Manje nedoslednosti ili problemi sa formatiranjem mogu biti pogodni za automatsku korekciju. Na primer, radnik za samooporavljanje podataka zadužen za popravku neispravnog JSON-a može da se nosi sa nedostajućim zarezima ili neizbeglim dvostrukim navodnicima bez značajnog menjanja značenja ili strukture podataka. Ove vrste grešaka se često mogu jednostavno ispraviti i imaju minimalan uticaj na ukupni integritet podataka.

Međutim, ozbiljnije greške koje fundamentalno menjaju značenje ili integritet podataka mogu zahtevati drugačiji pristup. U takvim slučajevima, automatske ispravke možda neće biti dovoljne, i ljudska intervencija može biti neophodna kako bi se osigurala tačnost i validnost podataka.

Ovde dolazi do izražaja koncept korišćenja same veštačke inteligencije za pomoć u određivanju ozbiljnosti greške. Korišćenjem mogućnosti AI modela, možemo dizajnirati samoisceljujuće radnike za podatke koji ne samo da ispravljaju greške, već i procenjuju ozbiljnost tih grešaka i donose informisane odluke o načinu njihovog rešavanja.

Na primer, razmotrimo samoisceljujućeg radnika za podatke zaduženog za ispravljanje nekonzistentnosti u podacima koji se ulivaju u bazu podataka klijenata. Radnik može biti dizajniran da analizira podatke i identifikuje potencijalne greške, kao što su nedostajuće ili konfliktne informacije. Međutim, umesto da automatski ispravlja sve greške, radnik može biti opremljen dodatnim pozivima alatima koji mu omogućavaju da označi ozbiljne greške za pregled od strane ljudi.

Evo primera kako se ovo može implementirati:

```ruby
class CustomerDataReviewer
  include Raix::ChatCompletion
  include Raix::FunctionDeclarations

  attr_accessor :customer

  function :flag_for_review, reason: { type: "string" } do |params|
    AdminNotifier.review_request(customer, params[:reason])
  end

  def initialize(customer)
    self.customer = customer
  end

  def call(customer_data)
    transcript << {
      system: "You are a customer data reviewer. Your task is to identify
        and correct inconsistencies in customer data.

        < additional instructions here... >

        If you encounter severe errors that require human review, use the
        `flag_for_review` tool to flag the data for manual intervention." }

    transcript << { user: customer.to_json }
    transcript << { assistant: "Reviewed/corrected data:\n```json\n" }

    self.stop = ["```"]

    chat_completion(json: true).then do |result|
      return if result.blank?

      customer.update(result)
    end
  end
end
```

U ovom primeru, CustomerDataHealer radnik je dizajniran da identifikuje i ispravi nedoslednosti u korisničkim podacima. Još jednom, koristimo Ograđivanje odgovora i Trbuhozborac da bismo dobili strukturirani izlaz. Važno je napomenuti da sistemska

direktiva radnika uključuje uputstva za korišćenje funkcije flag_for_review ako se naiđe na ozbiljne greške.

Kada radnik obrađuje korisničke podatke, analizira podatke i pokušava da ispravi sve nedoslednosti. Ako radnik utvrdi da su greške ozbiljne i zahtevaju ljudsku intervenciju, može koristiti alat flag_for_review da označi podatke i navede razlog za označavanje.

Metod chat_completion se poziva sa json: true da bi se ispravljeni korisnički podaci parsirali kao JSON. Ne postoji mogućnost za petlju nakon poziva funkcije, tako da će rezultat biti prazan ako je pozvana funkcija flag_for_review. U suprotnom, podaci o korisniku se ažuriraju pregledanim i potencijalno ispravljenim podacima.

Uključivanjem procene ozbiljnosti grešaka i opcije označavanja podataka za ljudski pregled, radnik za samozaceljujuće podatke postaje inteligentniji i prilagodljiviji. Može automatski da rešava manje greške, dok ozbiljnije greške prosleđuje ljudskim stručnjacima na ručnu intervenciju.

Specifični kriterijumi za određivanje ozbiljnosti grešaka mogu se definisati u direktivi radnika na osnovu domenskog znanja i poslovnih zahteva. Faktori kao što su uticaj na integritet podataka, mogućnost gubitka ili oštećenja podataka i posledice netačnih podataka mogu se uzeti u obzir prilikom procene ozbiljnosti.

Korišćenjem veštačke inteligencije za procenu ozbiljnosti grešaka i pružanjem opcija za ljudsku intervenciju, tehnike samozaceljujućih podataka mogu postići ravnotežu između automatizacije i održavanja tačnosti podataka. Ovaj pristup osigurava da se manje greške efikasno ispravljaju, dok ozbiljne greške dobijaju potrebnu pažnju i stručnost od ljudskih recenzenata.

Kompleksnost domena

Kada se razmatra primena tehnika samozaceljujućih podataka, važno je proceniti kompleksnost domena podataka i pravila koja regulišu njihovu strukturu i odnose. Kompleksnost domena može značajno uticati na efikasnost i izvodljivost automatizovanih pristupa ispravljanju podataka.

Tehnike samozaceljujućih podataka dobro funkcionišu kada podaci prate jasno definisane obrasce i ograničenja. U domenima gde je struktura podataka relativno jednostavna i odnosi između elemenata podataka su jasni, automatske ispravke se mogu primeniti sa visokim stepenom pouzdanosti. Na primer, ispravljanje problema sa formatiranjem ili primena osnovnih ograničenja tipa podataka često može biti efikasno rešeno pomoću radnika za samozaceljujuće podatke.

Međutim, kako se povećava kompleksnost domena podataka, rastu i izazovi povezani sa automatskim ispravljanjem podataka. U domenima sa složenom poslovnom logikom, kompleksnim odnosima između entiteta podataka ili domenski specifičnim pravilima i izuzecima, tehnike samozaceljujućih podataka možda neće uvek uhvatiti nijanse i mogu dovesti do neželjenih posledica.

Razmotrimo primer složenog domena: sistem za finansijsko trgovanje. U ovom domenu, podaci uključuju različite finansijske instrumente, tržišne podatke, pravila trgovanja i regulatorne zahteve. Odnosi između različitih elemenata podataka mogu biti složeni, a pravila koja regulišu validnost i konzistentnost podataka mogu biti vrlo specifična za domen.

U tako složenom domenu, radnik za samozaceljujuće podatke zadužen za ispravljanje nedoslednosti u podacima o trgovanju morao bi imati duboko razumevanje domenski specifičnih pravila i ograničenja. Morao bi uzeti u obzir faktore kao što su tržišni propisi, ograničenja trgovanja, izračuni rizika i procedure poravnanja. Automatske ispravke u ovom kontekstu možda neće uvek obuhvatiti punu kompleksnost domena i mogu nenamerno uvesti greške ili prekršiti domenski specifična pravila.

Za rešavanje izazova kompleksnosti domena, tehnike samozaceljujućih podataka mogu se unaprediti uključivanjem domenski specifičnog znanja i pravila u AI modele i radnike. Ovo se može postići kroz tehnike kao što su:

1. **Domenski specifična obuka**: AI modeli koji se koriste za samozaceljujuće podatke mogu biti usmereni ili čak fino podešeni na domenski specifičnim skupovima podataka koji obuhvataju složenost i pravila određenog domena.

Izlaganjem modela reprezentativnim podacima i scenarijima, oni mogu naučiti obrasce, ograničenja i izuzetke specifične za domen.

2. **Ograničenja zasnovana na pravilima**: Radnici za samozaceljujuće podatke mogu biti prošireni eksplicitnim ograničenjima zasnovanim na pravilima koja kodiraju domenski specifično znanje. Ova pravila mogu definisati domenski stručnjaci i integrisati ih u proces ispravljanja podataka. AI modeli tada mogu koristiti ova pravila za vođenje svojih odluka i osiguravanje usklađenosti sa domenski specifičnim zahtevima.
3. **Saradnja sa domenskim stručnjacima**: U složenim domenima, ključno je uključiti domenske stručnjake u dizajn i razvoj tehnika samozaceljujućih podataka. Domenski stručnjaci mogu pružiti vredne uvide u složenost podataka, poslovna pravila i potencijalne granične slučajeve. Njihovo znanje se može ugraditi u AI modele i radnike kako bi se poboljšala tačnost i pouzdanost automatskih ispravki podataka koristeći obrasce Čovek u petlji.
4. **Inkrementalni i iterativni pristup**: Kada se radi sa složenim domenima, često je korisno usvojiti inkrementalni i iterativni pristup samozaceljujućim podacima. Umesto pokušaja automatizacije ispravki za ceo domen odjednom, fokusirajte se na specifične poddomene ili kategorije podataka gde su pravila i ograničenja dobro shvaćena. Postepeno proširujte opseg tehnika samozaceljenja kako raste razumevanje domena i tehnike se pokažu efikasnim.

Uzimajući u obzir složenost domena podataka i ugrađivanje domenskog znanja u tehnike samozalečujućih podataka, možete postići ravnotežu između automatizacije i tačnosti. Važno je prepoznati da samozalečujući podaci nisu univerzalno rešenje i da pristup treba prilagoditi specifičnim zahtevima i izazovima svakog domena.

U složenim domenima, hibridni pristup koji kombinuje tehnike samozalečujućih podataka sa ljudskom ekspertizom i nadzorom može biti najefikasniji. Automatske korekcije mogu da se bave rutinskim i dobro definisanim slučajevima, dok se složeni scenariji ili izuzeci mogu označiti za ljudski pregled i intervenciju. Ovaj saradnički

pristup osigurava da se ostvare prednosti automatizacije uz održavanje neophodne kontrole i tačnosti u složenim domenima podataka.

Objašnjivost i Transparentnost

Objašnjivost se odnosi na sposobnost razumevanja i tumačenja rezonovanja iza odluka koje donose AI modeli, dok transparentnost podrazumeva obezbeđivanje jasne vidljivosti u proces korekcije podataka.

U mnogim kontekstima, izmene podataka moraju biti podložne reviziji i opravdane. Zainteresovane strane, uključujući poslovne korisnike, revizore i regulatorna tela, mogu zahtevati objašnjenja zašto su određene korekcije podataka napravljene i kako su AI modeli došli do tih odluka. Ovo je posebno važno u domenima gde tačnost i integritet podataka imaju značajne implikacije, kao što su finansije, zdravstvo i pravna pitanja.

Da bi se odgovorilo na potrebu za objašnjivošću i transparentnošću, tehnike samozalečujućih podataka treba da uključe mehanizme koji pružaju uvid u proces donošenja odluka AI modela. Ovo se može postići kroz različite pristupe:

1. **Lanac Razmišljanja**: Traženje od modela da objasni svoje razmišljanje "naglas" pre primene promena na podacima može omogućiti lakše razumevanje procesa donošenja odluka i može generisati objašnjenja razumljiva ljudima za napravljene korekcije. Kompromis je malo veća složenost u odvajanju objašnjenja od strukturiranog izlaza podataka, što se može rešiti…
2. **Generisanje Objašnjenja**: Radnici za samozalečenje podataka mogu biti opremljeni sposobnošću generisanja objašnjenja razumljivih ljudima za korekcije koje prave. Ovo se može postići tako što će se od modela tražiti da prikaže svoj proces donošenja odluka kao lako razumljiva objašnjenja *integrisana u same podatke*. Na primer, radnik za samozalečenje podataka mogao bi generisati izveštaj koji ističe specifične nedoslednosti u podacima koje je identifikovao, korekcije koje je primenio i obrazloženje iza tih korekcija.

3. **Važnost Karakteristika**: AI modelima se mogu dati uputstva sa informacijama o važnosti različitih karakteristika ili atributa u procesu korekcije podataka kao deo njihovih direktiva. Te direktive se zatim mogu pokazati ljudskim zainteresovanim stranama. Identifikovanjem ključnih faktora koji utiču na odluke modela, zainteresovane strane mogu steći uvid u rezonovanje iza korekcija i proceniti njihovu validnost.
4. **Beleženje i Revizija**: Implementacija sveobuhvatnih mehanizama za beleženje i reviziju je ključna za održavanje transparentnosti u procesu samozalečenja podataka. Svaka korekcija podataka koju naprave AI modeli treba da bude zabeležena, uključujući originalne podatke, korigovane podatke i specifične preduzete akcije. Ovaj revizorski trag omogućava retrospektivnu analizu i pruža jasan zapis o izmenama napravljenim na podacima.
5. **Pristup sa Čovekom u Petlji**: Uključivanje pristupa sa čovekom u petlji može poboljšati objašnjivost i transparentnost tehnika samozalečujućih podataka. Uključivanjem ljudskih stručnjaka u pregled i validaciju korekcija generisanih od strane AI-ja, organizacije mogu osigurati da su korekcije usklađene sa domenskim znanjem i poslovnim zahtevima. Ljudski nadzor dodaje dodatni sloj odgovornosti i omogućava identifikaciju potencijalnih pristrasnosti ili grešaka u AI modelima.
6. **Kontinuirano Praćenje i Evaluacija**: Redovno praćenje i evaluacija performansi tehnika samozalečujućih podataka je neophodno za održavanje transparentnosti i poverenja. Procenom tačnosti i efikasnosti AI modela tokom vremena, organizacije mogu identifikovati sva odstupanja ili anomalije i preduzeti korektivne mere. Kontinuirano praćenje pomaže da se osigura da proces samozalečenja podataka ostane pouzdan i usklađen sa željenim ishodima.

Objašnjivost i transparentnost su kritični faktori pri implementaciji tehnika samozalečujućih podataka. Pružanjem jasnih objašnjenja za korekcije podataka, održavanjem sveobuhvatnih revizorskih tragova i uključivanjem ljudskog nadzora, organizacije mogu izgraditi poverenje u proces samozalečenja podataka i osigurati da su izmene napravljene na podacima opravdane i usklađene sa poslovnim ciljevima.

Važno je postići ravnotežu između prednosti automatizacije i potrebe za transparentnošću. Iako tehnike samozalečujućih podataka mogu značajno poboljšati kvalitet podataka i efikasnost, one ne bi trebalo da budu na uštrb gubitka vidljivosti i kontrole nad procesom korekcije podataka. Dizajniranjem radnika za samozalečenje podataka sa fokusom na objašnjivost i transparentnost, organizacije mogu iskoristiti snagu AI-ja uz održavanje neophodnog nivoa odgovornosti i poverenja u podatke.

Nenamerne Posledice

Iako tehnike samozalečujućih podataka imaju za cilj poboljšanje kvaliteta i konzistentnosti podataka, ključno je biti svestan potencijalnih nenamernih posledica. Automatske korekcije, ako nisu pažljivo dizajnirane i praćene, mogu nenamerno izmeniti značenje ili kontekst podataka, što dovodi do problema u kasnijim fazama.

Jedan od primarnih rizika samozalečujućih podataka je uvođenje pristrasnosti ili grešaka u proces korekcije podataka. AI modeli, kao i bilo koji drugi softverski sistem, mogu biti podložni pristrasnostima prisutnim u podacima za obuku ili uvedenim kroz dizajn algoritama. Ako se ove pristrasnosti ne identifikuju i ne ublaže, one se mogu propagirati kroz proces samozalečenja podataka i rezultirati iskrivljenim ili netačnim modifikacijama podataka.

Na primer, razmotrimo samoisceljujućeg radnika za podatke zaduženog za ispravljanje nedoslednosti u demografskim podacima kupaca. Ako je AI model naučio pristrasnosti iz istorijskih podataka, kao što je povezivanje određenih zanimanja ili nivoa prihoda sa specifičnim polovima ili etničkim pripadnostima, može praviti pogrešne pretpostavke i modifikovati podatke na način koji pojačava te pristrasnosti. Ovo može dovesti do netačnih profila kupaca, pogrešnih poslovnih odluka i potencijalno diskriminatornih ishoda.

Još jedna potencijalna neželjena posledica je gubitak vrednih informacija ili konteksta tokom procesa ispravljanja podataka. Tehnike samoisceljujućih podataka se često fokusiraju na standardizaciju i normalizaciju podataka kako bi se osigurala

konzistentnost. Međutim, u nekim slučajevima, originalni podaci mogu sadržati nijanse, izuzetke ili kontekstualne informacije koje su važne za razumevanje celokupne slike. Automatizovane ispravke koje slepo nameću standardizaciju mogu nenamerno ukloniti ili zamagliti ove vredne informacije.

Na primer, zamislite samoisceljujućeg radnika za podatke odgovornog za ispravljanje nedoslednosti u medicinskim kartonima. Ako radnik naiđe na medicinsku istoriju pacijenta sa retkim stanjem ili neuobičajenim planom lečenja, može pokušati da normalizuje podatke kako bi se uklopili u češći obrazac. Međutim, čineći to, može izgubiti specifične detalje i kontekst koji su ključni za tačno predstavljanje jedinstvene situacije pacijenta. Ovaj gubitak informacija može imati ozbiljne implikacije za negu pacijenta i donošenje medicinskih odluka.

Da bi se ublažili rizici od neželjenih posledica, neophodno je zauzeti proaktivan pristup pri dizajniranju i implementaciji tehnika samoisceljujućih podataka:

1. **Temeljno testiranje i validacija**: Pre implementacije samoisceljujućih radnika za podatke u produkciji, ključno je temeljno testirati i validirati njihovo ponašanje u različitim scenarijima. Ovo uključuje testiranje sa reprezentativnim skupovima podataka koji pokrivaju različite granične slučajeve, izuzetke i potencijalne pristrasnosti. Rigorozno testiranje pomaže u identifikovanju i rešavanju neželjenih posledica pre nego što utiču na podatke u stvarnom svetu.
2. **Kontinuirano praćenje i evaluacija**: Implementacija mehanizama za kontinuirano praćenje i evaluaciju je ključna za otkrivanje i ublažavanje neželjenih posledica tokom vremena. Redovno pregledanje ishoda procesa samoisceljujućih podataka, analiziranje uticaja na nizvodne sisteme i donošenje odluka, kao i prikupljanje povratnih informacija od zainteresovanih strana može pomoći u identifikovanju štetnih efekata i pokretanju pravovremenih korektivnih akcija. Ako vaša organizacija ima operativne kontrolne table, dodavanje jasno vidljivih metrika vezanih za automatizovane promene podataka je verovatno dobra ideja. Dodavanje alarma povezanih sa velikim odstupanjima od normalne

aktivnosti promene podataka je verovatno još bolja ideja!

3. **Ljudski nadzor i intervencija**: Održavanje ljudskog nadzora i mogućnosti intervencije u procesu samoisceljujućih podataka je ključno. Iako automatizacija može značajno poboljšati efikasnost, važno je da ljudski stručnjaci pregledaju i validiraju ispravke koje prave AI modeli, posebno u kritičnim ili osetljivim domenima. Ljudska procena i stručnost u domenu mogu pomoći u identifikovanju i rešavanju neželjenih posledica koje se mogu pojaviti.

4. **Objašnjiva veštačka inteligencija (XAI) i transparentnost**: Kao što je diskutovano u prethodnom pododeljku, uključivanje tehnika objašnjive veštačke inteligencije i osiguravanje transparentnosti u procesu samoisceljujućih podataka može pomoći u ublažavanju neželjenih posledica. Pružanjem jasnih objašnjenja za ispravke podataka i održavanjem sveobuhvatnih revizorskih tragova, organizacije mogu bolje razumeti i pratiti rezonovanje iza modifikacija koje prave AI modeli.
5. **Inkrementalni i iterativni pristup**: Usvajanje inkrementalnog i iterativnog pristupa samoisceljujućim podacima može pomoći u minimiziranju rizika od neželjenih posledica. Umesto primene automatizovanih ispravki na celom skupu podataka odjednom, počnite sa podskupom podataka i postepeno proširujte opseg kako se tehnike pokažu efikasnim i pouzdanim. Ovo omogućava pažljivo praćenje i prilagođavanje tokom procesa, smanjujući uticaj neželjenih posledica.
6. **Saradnja i povratne informacije**: Uključivanje zainteresovanih strana iz različitih domena i podsticanje saradnje i povratnih informacija tokom procesa samoisceljujućih podataka može pomoći u identifikovanju i rešavanju neželjenih posledica. Redovno traženje inputa od stručnjaka iz domena, korisnika podataka i krajnjih korisnika može pružiti vredne uvide u stvarni uticaj ispravki podataka i istaći probleme koji su možda bili previđeni.

Proaktivnim rešavanjem rizika od neželjenih posledica i implementacijom odgovarajućih zaštitnih mera, organizacije mogu iskoristiti prednosti tehnika samoisceljujućih podataka uz minimiziranje potencijalnih štetnih efekata. Važno

je pristupiti samoisceljujućim podacima kao iterativnom i kolaborativnom procesu, kontinuirano prateći, evaluirajući i usavršavajući tehnike kako bi se osiguralo da su usklađene sa željenim ishodima i održavaju integritet i pouzdanost podataka.

Kada se razmatra upotreba obrazaca samoisceljujućih podataka, neophodno je pažljivo proceniti ove faktore i odvagnuti prednosti u odnosu na potencijalne rizike i ograničenja. U nekim slučajevima, hibridni pristup koji kombinuje automatizovane ispravke sa ljudskim nadzorom i intervencijom može biti najprikladnije rešenje.

Takođe je vredno napomenuti da tehnike samoisceljujućih podataka ne bi trebalo posmatrati kao zamenu za robusnu validaciju podataka, sanitizaciju unosa i mehanizme za rukovanje greškama. Ove fundamentalne prakse ostaju kritične za osiguravanje integriteta i bezbednosti podataka. Samoisceljujuće podatke treba posmatrati kao komplementarni pristup koji može proširiti i poboljšati ove postojeće mere.

Na kraju, odluka o upotrebi obrazaca samoisceljujućih podataka zavisi od specifičnih zahteva, ograničenja i prioriteta vaše aplikacije. Pažljivim razmatranjem gore navedenih faktora i njihovim usklađivanjem sa ciljevima i arhitekturom vaše aplikacije, možete doneti informisane odluke o tome kada i kako efikasno koristiti tehnike samoisceljujućih podataka.

Kontekstualno generisanje sadržaja

Obrasci kontekstualnog generisanja sadržaja koriste moć velikih jezičkih modela (LLM) za generisanje dinamičkog i kontekstualno specifičnog sadržaja unutar aplikacija. Ova kategorija obrazaca prepoznaje važnost isporuke personalizovanog i relevantnog sadržaja korisnicima na osnovu njihovih specifičnih potreba, preferenci, pa čak i prethodnih i trenutnih interakcija sa aplikacijom.

> U kontekstu ovog pristupa, "sadržaj" se odnosi kako na primarni sadržaj (tj. blog postove, članke, itd.) tako i na meta-sadržaj, poput preporuka za primarni sadržaj.

Obrasci kontekstualnog generisanja sadržaja mogu igrati ključnu ulogu u poboljšanju nivoa angažovanja korisnika, pružanju prilagođenih iskustava i automatizaciji zadataka kreiranja sadržaja kako za vas tako i za vaše korisnike. Koristeći obrasce koje

opisujemo u ovom poglavlju, možete kreirati aplikacije koje dinamički generišu sadržaj, prilagođavajući se kontekstu i ulaznim podacima u realnom vremenu.

Obrasci funkcionišu integrisanjem LLM-ova u izlaze aplikacije, od korisničkog interfejsa (ponekad nazvanog "chrome"), do imejlova i drugih oblika obaveštenja, kao i bilo kojih procesa generisanja sadržaja.

Kada korisnik komunicira sa aplikacijom ili pokrene specifičan zahtev za sadržajem, aplikacija hvata relevantni kontekst, kao što su korisničke preference, prethodne interakcije ili specifični upiti. Ove kontekstualne informacije se zatim prosleđuju LLM-u, zajedno sa svim potrebnim šablonima ili smernicama i koriste se za proizvodnju tekstualnog izlaza koji bi inače morao biti ili hardkodiran, sačuvan u bazi podataka ili algoritmički generisan.

Sadržaj generisan pomoću LLM-a može imati različite forme, kao što su personalizovane preporuke, dinamički opisi proizvoda, prilagođeni odgovori na imejlove, pa čak i celokupni članci ili blog postovi. Jedna od najradikalnijih upotreba ovog sadržaja koju sam započeo pre više od godinu dana je dinamičko generisanje UI elemenata poput oznaka formulara, objašnjenja i drugih vrsta tekstova za pojašnjenje.

Personalizacija

Jedna od ključnih prednosti obrazaca kontekstualnog generisanja sadržaja je mogućnost pružanja visoko personalizovanih iskustava korisnicima. Generisanjem sadržaja na osnovu konteksta specifičnog za korisnika, ovi obrasci omogućavaju aplikacijama da prilagode sadržaj individualnim interesima, preferencama i interakcijama korisnika.

Personalizacija prevazilazi jednostavno ubacivanje korisničkog imena u generički sadržaj. Ona uključuje korišćenje bogatog konteksta dostupnog o svakom korisniku za generisanje sadržaja koji rezonuje sa njihovim specifičnim potrebama i željama. Ovaj kontekst može uključivati širok spektar faktora, kao što su:

1. **Informacije korisničkog profila**: Na najopštijem nivou primene ove tehnike, demografski podaci, interesovanja, preference i drugi atributi profila mogu se koristiti za generisanje sadržaja koji se usklađuje sa korisnikovim poreklom i karakteristikama.
2. **Podaci o ponašanju**: Prethodne interakcije korisnika sa aplikacijom, kao što su pregledane stranice, kliknuti linkovi ili kupljeni proizvodi, mogu pružiti vredne uvide u njihovo ponašanje i interesovanja. Ovi podaci se mogu koristiti za generisanje predloga sadržaja koji odražava njihove obrasce angažovanja i predviđa njihove buduće potrebe.
3. **Kontekstualni faktori**: Trenutni kontekst korisnika, kao što su njihova lokacija, uređaj, doba dana, pa čak i vremenske prilike, mogu uticati na proces generisanja sadržaja. Na primer, aplikacija za putovanja bi mogla imati AI radnika koji je u stanju da generiše personalizovane preporuke na osnovu trenutne lokacije korisnika i preovlađujućih vremenskih uslova.

Korišćenjem ovih kontekstualnih faktora, obrasci kontekstualnog generisanja sadržaja omogućavaju aplikacijama da isporuče sadržaj koji deluje kao da je napravljen posebno za svakog pojedinačnog korisnika. Ovaj nivo personalizacije ima nekoliko značajnih prednosti:

1. **Povećan angažman**: Personalizovani sadržaj privlači pažnju korisnika i održava njihovu angažovanost sa aplikacijom. Kada korisnici osete da je sadržaj relevantan i da direktno odgovara njihovim potrebama, veća je verovatnoća da će provesti više vremena u interakciji sa aplikacijom i istraživanju njenih funkcija.
2. **Poboljšano zadovoljstvo korisnika**: Personalizovani sadržaj pokazuje da aplikacija razume i brine o jedinstvenim zahtevima korisnika. Pružanjem sadržaja koji je koristan, informativan i usklađen sa njihovim interesovanjima, aplikacija može povećati zadovoljstvo korisnika i izgraditi snažniju vezu sa svojim korisnicima.

3. **Više stope konverzije**: U kontekstu e-trgovine ili marketing aplikacija, personalizovani sadržaj može značajno uticati na stope konverzije. Predstavljanjem proizvoda, ponuda ili preporuka koje su prilagođene njihovim preferencama i ponašanju, aplikacija može povećati verovatnoću da će korisnici preduzeti željene akcije, kao što su kupovina ili registracija za uslugu.

Produktivnost

Obrasci kontekstualnog generisanja sadržaja mogu značajno povećati određene vrste produktivnosti smanjujući potrebu za ručnim generisanjem i uređivanjem sadržaja u kreativnim procesima. Korišćenjem moći LLM-ova, možete generisati kvalitetan sadržaj u velikim razmerama, štedeći vreme i trud koji bi vaši kreatori sadržaja i programeri inače morali da utroše na zamoran ručni rad.

Tradicionalno, kreatori sadržaja moraju da istražuju, pišu, uređuju i formatiraju sadržaj kako bi osigurali da ispunjava zahteve aplikacije i očekivanja korisnika. Ovaj proces može biti vremenski zahtevan i resursno intenzivan, posebno kako obim sadržaja raste.

Međutim, sa obrascima kontekstualnog generisanja sadržaja, proces kreiranja sadržaja može biti u velikoj meri automatizovan. Veliki jezički modeli mogu generisati koherentan, gramatički ispravan i kontekstualno relevantan sadržaj na osnovu datih promptova i smernica. Ova automatizacija nudi nekoliko prednosti u pogledu produktivnosti:

1. **Smanjen ručni rad**: Delegiranjem zadataka generisanja sadržaja velikim jezičkim modelima, kreatori sadržaja se mogu fokusirati na zadatke višeg nivoa kao što su strategija sadržaja, ideacija i osiguranje kvaliteta. Oni mogu pružiti potreban kontekst, šablone i smernice jezičkom modelu i pustiti ga da se bavi stvarnim generisanjem sadržaja. Ovo smanjuje ručni rad potreban za pisanje i uređivanje, omogućavajući kreatorima sadržaja da budu produktivniji i efikasniji.

2. **Brže kreiranje sadržaja**: Veliki jezički modeli mogu generisati sadržaj mnogo brže od ljudskih pisaca. Sa pravim promptovima i smernicama, jezički model može proizvesti više delova sadržaja u roku od nekoliko sekundi ili minuta. Ova brzina omogućava aplikacijama da generišu sadržaj mnogo bržim tempom, držeći korak sa zahtevima korisnika i stalno promenljivim digitalnim okruženjem.

> Da li brže kreiranje sadržaja vodi do situacije "tragedije zajedničkog dobra" gde se internet guši u sadržaju koji niko ne čita. Nažalost, sumnjam da je odgovor potvrdan.

3. **Konzistentnost i kvalitet**: Veliki jezički modeli mogu trivijalno revidirati sadržaj tako da bude konzistentan u stilu, tonu i kvalitetu. Uz jasne smernice i primere, određene vrste aplikacija (tj. redakcije, PR, itd.) mogu osigurati da njihov sadržaj koji generišu ljudi bude usklađen sa glasom brenda i zadovoljava željene standarde kvaliteta. Ova konzistentnost smanjuje potrebu za obimnim uređivanjem i revizijama, štedeći vreme i trud u procesu kreiranja sadržaja.
4. **Iteracija i optimizacija**: Obrasci kontekstualnog generisanja sadržaja omogućavaju brzu iteraciju i optimizaciju sadržaja. Podešavanjem promptova, šablona ili smernica datih jezičkom modelu, vaše aplikacije mogu brzo generisati varijacije sadržaja i testirati različite pristupe na automatizovan način koji nikada ranije nije bio moguć. Ovaj iterativni proces omogućava brže eksperimentisanje i usavršavanje strategija sadržaja, što vremenom dovodi do efektivnijeg i angažovanijeg sadržaja. Ova konkretna tehnika može biti potpuna prekretnica za aplikacije kao što je e-trgovina koje žive ili umiru na osnovu stope napuštanja i angažovanja

Važno je napomenuti da iako obrasci kontekstualnog generisanja sadržaja mogu značajno povećati produktivnost, oni ne eliminišu u potpunosti potrebu za ljudskim učešćem. Kreatori sadržaja i urednici i dalje igraju ključnu ulogu u definisanju celokupne strategije sadržaja, pružanju smernica jezičkom modelu i osiguravanju kvaliteta i prikladnosti generisanog sadržaja.

Automatizacijom više repetitivnih i vremenski zahtevnih aspekata kreiranja sadržaja, obrasci kontekstualnog generisanja sadržaja oslobađaju dragoceno ljudsko vreme i resurse koji se mogu preusmeriti na zadatke više vrednosti. Ova povećana produktivnost vam omogućava da isporučite personalizovaniji i angažovaniji sadržaj korisnicima uz optimizaciju radnih tokova kreiranja sadržaja.

Brza iteracija i eksperimentisanje

Obrasci kontekstualnog generisanja sadržaja vam omogućavaju da brzo iterirate i eksperimentišete sa različitim varijacijama sadržaja, omogućavajući bržu optimizaciju i usavršavanje vaše strategije sadržaja. Možete generisati više verzija sadržaja u roku od nekoliko sekundi, jednostavnim podešavanjem konteksta, šablona ili smernica datih modelu.

Ova mogućnost brze iteracije nudi nekoliko ključnih prednosti:

1. **Testiranje i optimizacija**: Sa mogućnošću brzog generisanja varijacija sadržaja, lako možete testirati različite pristupe i meriti njihovu efikasnost. Na primer, možete generisati više verzija opisa proizvoda ili marketinške poruke, svaku prilagođenu određenom segmentu korisnika ili kontekstu. Analiziranjem metrika angažovanja korisnika, kao što su stope klikova ili stope konverzije, možete identifikovati najefikasnije varijacije sadržaja i optimizovati vašu strategiju sadržaja u skladu s tim.

2. **A/B testiranje**: Obrasci kontekstualnog generisanja sadržaja omogućavaju besprekorno A/B testiranje sadržaja. Možete generisati dve ili više varijacija sadržaja i nasumično ih servirati različitim grupama korisnika. Poređenjem performansi svake varijacije, možete utvrditi koji sadržaj najbolje rezonuje sa vašom ciljnom publikom. Ovaj pristup zasnovan na podacima vam omogućava da donosite informisane odluke i kontinuirano usavršavate svoj sadržaj kako biste maksimizirali angažovanje korisnika i postigli željene rezultate.
3. **Eksperimenti sa personalizacijom**: Brza iteracija i eksperimentisanje su posebno vredni kada je reč o personalizaciji. Sa obrascima kontekstualnog generisanja sadržaja, možete brzo generisati personalizovane varijacije sadržaja zasnovane na različitim segmentima korisnika, preferencijama ili ponašanjima. Eksperimentisanjem sa različitim strategijama personalizacije, možete identifikovati najefikasnije pristupe za angažovanje pojedinačnih korisnika i pružanje prilagođenih iskustava.
4. **Prilagođavanje promenljivim trendovima**: Mogućnost brze iteracije i eksperimentisanja vam omogućava da ostanete agilni i prilagodite se promenljivim trendovima i preferencijama korisnika. Kako se pojavljuju nove teme, ključne reči ili ponašanja korisnika, možete brzo generisati sadržaj koji je usklađen sa ovim trendovima. Kontinuiranim eksperimentisanjem i usavršavanjem vašeg sadržaja možete ostati relevantni i održati konkurentsku prednost u digitalnom okruženju koje se neprestano razvija.
5. **Ekonomično eksperimentisanje**: Tradicionalno eksperimentisanje sa sadržajem često zahteva značajno vreme i resurse, jer kreatori sadržaja moraju ručno da razvijaju i testiraju različite varijacije. Međutim, sa obrascima Kontekstualnog generisanja sadržaja, troškovi eksperimentisanja su značajno smanjeni. Veliki jezički modeli mogu brzo generisati varijacije sadržaja u većem obimu, omogućavajući vam da istražite širok spektar ideja i pristupa bez značajnih troškova.

Da biste maksimalno iskoristili brzu iteraciju i eksperimentisanje, važno je imati dobro

definisan eksperimentalni radni okvir. Ovaj okvir treba da uključuje:

- Jasne ciljeve i hipoteze za svaki eksperiment
- Odgovarajuće metrike i mehanizme praćenja za merenje performansi sadržaja
- Strategije segmentacije i ciljanja kako bi se osiguralo da odgovarajuće varijacije sadržaja budu isporučene pravim korisnicima
- Alate za analizu i izveštavanje za izvlačenje uvida iz eksperimentalnih podataka
- Proces za uključivanje naučenog i optimizacija u vašu strategiju sadržaja

Prihvatanjem brze iteracije i eksperimentisanja, možete kontinuirano usavršavati i optimizovati vaš sadržaj, osiguravajući da ostane privlačan, relevantan i efikasan u postizanju ciljeva vaše aplikacije. Ovaj agilni pristup kreiranju sadržaja vam omogućava da budete korak ispred i pružite izuzetno korisničko iskustvo.

Skalabilnost i efikasnost

Kako aplikacije rastu i potražnja za personalizovanim sadržajem se povećava, obrasci kontekstualnog generisanja sadržaja omogućavaju efikasno skaliranje kreiranja sadržaja. Veliki jezički modeli mogu istovremeno generisati sadržaj za veliki broj korisnika i konteksta, bez potrebe za proporcionalnim povećanjem ljudskih resursa. Ova skalabilnost omogućava aplikacijama da isporuče personalizovana iskustva rastućoj bazi korisnika bez opterećenja njihovih mogućnosti kreiranja sadržaja.

Imajte na umu da se kontekstualno generisanje sadržaja može efikasno koristiti za internacionalizaciju vaše aplikacije "u hodu". Zapravo, to je tačno ono što sam uradio koristeći svoj Instant18n Gem za isporuku Olympie na više od pola desetine jezika, iako nismo ni godinu dana stari.

AI podržana lokalizacija

Ako mi dozvolite da se malo pohvalim, mislim da je moja Instant18n biblioteka za Rails aplikacije revolucionarni primer obrasca "Kontekstualnog generisanja sadržaja" na delu, pokazujući transformativni potencijal veštačke inteligencije u razvoju aplikacija. Ovaj gem koristi snagu OpenAI-jevog GPT velikog jezičkog modela da revolucionira način na koji se internacionalizacija i lokalizacija obrađuju u Rails aplikacijama.

Tradicionalno, internacionalizacija Rails aplikacije uključuje ručno definisanje ključeva za prevođenje i obezbeđivanje odgovarajućih prevoda za svaki podržani jezik. Ovaj proces može biti dugotrajan, zahtevan u pogledu resursa i sklon nedoslednostima. Međutim, sa Instant18n gem-om, paradigma lokalizacije je potpuno redefinisana.

Integracijom velikog jezičkog modela, Instant18n gem vam omogućava da generišete prevode u hodu, na osnovu konteksta i značenja teksta. Umesto oslanjanja na unapred definisane ključeve za prevođenje i statičke prevode, gem dinamički prevodi tekst koristeći snagu veštačke inteligencije. Ovaj pristup nudi nekoliko ključnih prednosti:

1. **Besprekorna lokalizacija**: Sa Instant18n gem-om, programeri više ne moraju ručno da definišu i održavaju datoteke sa prevodima za svaki podržani jezik. Gem automatski generiše prevode na osnovu datog teksta i željenog ciljnog jezika, čineći proces lokalizacije jednostavnim i besprekornim.
2. **Kontekstualna preciznost**: AI može dobiti dovoljno konteksta da shvati nijanse teksta koji se prevodi. Može uzeti u obzir okolni kontekst, idiome i kulturološke reference kako bi generisao prevode koji su precizni, prirodni i kontekstualno prikladni.
3. **Opsežna jezička podrška**: Instant18n gem koristi ogromno znanje i lingvističke sposobnosti GPT-a, omogućavajući prevode na širok spektar jezika. Od uobičajenih jezika poput španskog i francuskog do opskurnijih ili izmišljenih jezika poput klingonskog i vilovnjačkog, gem može da se nosi sa širokim spektrom prevodilačkih zahteva.

4. **Fleksibilnost i kreativnost**: Gem prevazilazi tradicionalne jezičke prevode i omogućava kreativne i nekonvencionalne opcije lokalizacije. Programeri mogu prevoditi tekst u različite stilove, dijalekte, pa čak i izmišljene jezike, otvarajući nove mogućnosti za jedinstvena korisnička iskustva i privlačan sadržaj.
5. **Optimizacija performansi**: Instant18n gem uključuje mehanizme keširanja za poboljšanje performansi i smanjenje opterećenja kod ponovljenih prevoda. Prevedeni tekst se kešira, omogućavajući da se naknadni zahtevi za istim prevodom brzo isporuče bez potrebe za redundantnim API pozivima.

Instant18n gem predstavlja primer snage obrasca "Kontekstualnog generisanja sadržaja" korišćenjem veštačke inteligencije za dinamičko generisanje lokalizovanog sadržaja. On pokazuje kako se veštačka inteligencija može integrisati u osnovnu funkcionalnost Rails aplikacije, transformišući način na koji programeri pristupaju internacionalizaciji i lokalizaciji.

Eliminisanjem potrebe za ručnim upravljanjem prevoda i omogućavanjem prevođenja u realnom vremenu na osnovu konteksta, Instant18n gem štedi programerima značajno vreme i trud. Omogućava im da se fokusiraju na izgradnju osnovnih funkcionalnosti svoje aplikacije, dok se istovremeno osigurava da se aspekt lokalizacije odvija besprekorno i precizno.

Važnost korisničkog testiranja i povratnih informacija

Na kraju, uvek imajte na umu važnost korisničkog testiranja i povratnih informacija. Ključno je potvrditi da generisanje kontekstualnog sadržaja ispunjava očekivanja korisnika i usklađeno je sa ciljevima aplikacije. Kontinuirano unapređujte i usavršavajte generisani sadržaj na osnovu korisničkih uvida i analitike. Ako generišete dinamički sadržaj u velikom obimu koji bi bilo nemoguće ručno validirati od strane vas i vašeg tima, razmotrite dodavanje mehanizama za povratne informacije koji omogućavaju korisnicima da prijave sadržaj koji je čudan ili pogrešan, zajedno sa objašnjenjem zašto.

Te dragocene povratne informacije mogu čak biti prosleđene AI programu zaduženom za prilagođavanje komponente koja je generisala sadržaj!

Generativni UI

Pažnja je danas toliko dragocena da efektivno angažovanje korisnika zahteva softverska iskustva koja nisu samo besprekorna i intuitivna, već i visoko personalizovana prema individualnim potrebama, preferencijama i kontekstima. Kao rezultat toga, dizajneri i programeri se sve više suočavaju sa izazovom kreiranja korisničkih interfejsa koji se mogu prilagoditi i odgovoriti jedinstvenim zahtevima svakog korisnika *u velikim razmerama.*

Generativni UI (GenUI) je zaista revolucionarni pristup dizajnu korisničkog interfejsa koji koristi moć velikih jezičkih modela (LLM) za kreiranje visoko personalizovanih i dinamičkih korisničkih iskustava u realnom vremenu. Želeo sam da vam u ovoj knjizi bar dam uvod u GenUI, jer verujem da je to jedna od najplodnijih prilika koje trenutno postoje u domenu dizajna aplikacija i radnih okvira. Ubeđen sam da će se u ovoj posebnoj niši pojaviti desetine ili više novih uspešnih komercijalnih i projekata otvorenog koda.

U svojoj suštini, GenUI kombinuje principe Kontekstualnog generisanja sadržaja sa naprednim AI tehnikama za dinamičko generisanje elemenata korisničkog interfejsa, kao što su tekst, slike i rasporedi, na osnovu dubokog razumevanja korisničkog konteksta, preferencija i ciljeva. GenUI omogućava dizajnerima i programerima da kreiraju interfejse koji se prilagođavaju i razvijaju kao odgovor na korisničke interakcije, pružajući nivo personalizacije koji ranije nije bio dostižan.

GenUI predstavlja fundamentalnu promenu u načinu na koji pristupamo dizajnu korisničkog interfejsa. Umesto dizajniranja za mase, GenUI nam omogućava da dizajniramo za pojedinca. Personalizovani sadržaj i interfejsi imaju potencijal da kreiraju korisnička iskustva koja rezonuju sa svakim korisnikom na dubljem nivou, povećavajući angažovanje, zadovoljstvo i lojalnost.

Kao najsavremenija tehnika, prelazak na GenUI je pun konceptualnih i praktičnih izazova. Integracija AI u proces dizajna, osiguravanje da generisani interfejsi nisu samo personalizovani već i upotrebljivi, pristupačni i usklađeni sa celokupnim brendom i korisničkim iskustvom, sve su to izazovi koji čine GenUI poduhvatom za malobrojne, ne za mnoge. Dodatno, uključivanje AI pokreće pitanja o privatnosti podataka, transparentnosti i čak etičkim implikacijama.

Uprkos izazovima, personalizovana iskustva u velikim razmerama imaju moć da potpuno transformišu način na koji interagujemo sa digitalnim proizvodima i uslugama. To otvara mogućnosti za kreiranje inkluzivnih i pristupačnih interfejsa koji odgovaraju različitim potrebama korisnika, bez obzira na njihove sposobnosti, poreklo ili preferencije.

U ovom poglavlju, istražićemo koncept GenUI-ja, ispitujući neke od definišućih karakteristika, ključnih prednosti i potencijalnih izazova. Počinjemo razmatranjem najosnovnijeg i najpristupačnijeg oblika GenUI-ja: generisanje tekstualnog sadržaja za inače tradicionalno dizajnirane i implementirane korisničke interfejse.

Generisanje teksta za korisničke interfejse

Tekstualni elementi koji postoje u chrome-u vaše aplikacije, kao što su oznake formulara, opisi alata i objašnjavajući tekst, obično su hardkodirani u šablone ili UI komponente, pružajući konzistentno ali generičko iskustvo za sve korisnike. Koristeći obrasce kontekstualnog generisanja sadržaja, možete transformisati ove statičke elemente u dinamičke, kontekstualno svesne i personalizovane komponente.

Personalizovani formulari

Formulari su sveprisutni deo web i mobilnih aplikacija, služeći kao primarno sredstvo za prikupljanje korisničkog unosa. Međutim, tradicionalni formulari često predstavljaju generičko i bezlično iskustvo, sa standardnim oznakama i poljima koja se ne moraju uvek poklapati sa specifičnim kontekstom ili potrebama korisnika. Korisnici će verovatnije popuniti formulare koji deluju prilagođeni njihovim potrebama i preferencijama, što dovodi do većih stopa konverzije i zadovoljstva korisnika.

Međutim, važno je postići ravnotežu između personalizacije i konzistentnosti. Iako prilagođavanje formulara individualnim korisnicima može biti korisno, ključno je održati nivo poznatosti i predvidljivosti. Korisnici bi i dalje trebalo da mogu lako prepoznati i navigirati kroz formulare, čak i sa personalizovanim elementima.

Evo nekoliko ideja za personalizovane formulare kao inspiracija:

Kontekstualni predlozi polja

GenUI može analizirati prethodne interakcije korisnika, preferencije i podatke kako bi pružio inteligentne predloge polja kao predviđanja. Na primer, ako je korisnik prethodno uneo svoju adresu za dostavu, formular može automatski popuniti relevantna polja njihovim sačuvanim informacijama. Ovo ne samo da štedi vreme već i pokazuje da aplikacija razume i pamti korisničke preferencije.

Čekajte malo, zar se ova tehnika ne bi mogla primeniti i bez upotrebe veštačke inteligencije? Naravno, ali lepota implementacije ovakve funkcionalnosti pomoću veštačke inteligencije je dvostruka: 1) koliko je lako implementirati je i 2) koliko je otporna na promene i evoluciju vašeg korisničkog interfejsa tokom vremena.

Hajde da na brzinu napravimo servis za naš teoretski sistem za obradu porudžbina, koji će pokušati da proaktivno popuni odgovarajuću adresu za dostavu umesto korisnika.

```
class OrderShippingAddressSubscriber
  include Raix::ChatCompletion

  attr_accessor :order

  delegate :customer, to: :order

  DIRECTIVE = "You are a smart order processing assistant. Given the
  customer's order history, guess the most likely shipping address
  for the current order."

  def order_created(order)
    return unless order.pending? && order.shipping_address.blank?

    self.order = order

    transcript.clear
    transcript << { system: DIRECTIVE }
    transcript << { user: "Order History: #{order_history.to_json}" }
    transcript << { user: "Current Order: #{order.to_json}" }

    response = chat_completion
    apply_predicted_shipping_address(order, response)
  end

  private

  def apply_predicted_shipping_address(order, response)
    # extract the shipping address from the response...
    # ...and assume there's some sort of live update of the address fields
    order.update(shipping_address:)
  end
```

```ruby

  def order_history
    customer.orders.successful.limit(100).map do |order|
      {
        date: order.date,
        description: order.description,
        shipping_address: order.shipping_address
      }
    end
  end
end
```

Ovaj primer je veoma pojednostavljen, ali bi trebalo da funkcioniše u većini slučajeva. Ideja je da pustimo veštačku inteligenciju da pogađa na isti način kao što bi to činio čovek. Da bih jasnije objasnio o čemu govorim, razmotrimo neke uzorne podatke:

```
Order History:
[
  {"date": "2024-01-03", "description": "garden soil mix",
   "shipping_address": "123 Country Lane, Rural Town"},
  {"date": "2024-01-15", "description": "hardcover fiction novels",
   "shipping_address": "456 City Apt, Metroville"},
  {"date": "2024-01-22", "description": "baby diapers", "shipping_address":
   "789 Suburb St, Quietville"},
  {"date": "2024-02-01", "description": "organic vegetables",
   "shipping_address": "123 Country Lane, Rural Town"},
  {"date": "2024-02-17", "description": "mystery thriller book set",
   "shipping_address": "456 City Apt, Metroville"},
  {"date": "2024-02-25", "description": "baby wipes",
   "shipping_address": "789 Suburb St, Quietville"},
  {"date": "2024-03-05", "description": "flower seeds",
   "shipping_address": "123 Country Lane, Rural Town"},
  {"date": "2024-03-20", "description": "biographies",
   "shipping_address": "456 City Apt, Metroville"},
  {"date": "2024-03-30", "description": "baby formula",
   "shipping_address": "789 Suburb St, Quietville"},
  {"date": "2024-04-12", "description": "lawn fertilizer",
   "shipping_address": "123 Country Lane, Rural Town"},
  {"date": "2024-04-22", "description": "science fiction novels",
   "shipping_address": "456 City Apt, Metroville"},
```

```
  {"date": "2024-05-02", "description": "infant toys",
   "shipping_address": "789 Suburb St, Quietville"},
  {"date": "2024-05-14", "description": "outdoor grill",
   "shipping_address": "123 Country Lane, Rural Town"},
  {"date": "2024-05-29", "description": "literary classics",
  "shipping_address": "456 City Apt, Metroville"},
  {"date": "2024-06-11", "description": "baby clothes",
   "shipping_address": "789 Suburb St, Quietville"},
  {"date": "2024-07-01", "description": "watering can",
   "shipping_address": "123 Country Lane, Rural Town"},
  {"date": "2024-07-18", "description": "non-fiction essays",
  "shipping_address": "456 City Apt, Metroville"},
  {"date": "2024-07-28", "description": "baby bath items",
  "shipping_address": "789 Suburb St, Quietville"},
  {"date": "2024-08-09", "description": "herb garden kit",
  "shipping_address": "123 Country Lane, Rural Town"},
  {"date": "2024-08-24", "description": "children's books",
  "shipping_address": "456 City Apt, Metroville"}
]
```

Da li ste primetili obrazac u podacima? Garantujem vam da je ovo jednostavno za VJM. Da bismo to demonstrirali, pitaćemo GPT-4 koja je najverovatnija adresa za isporuku za “toplomer”.

```
From the order history you've provided, it looks like the purchases are
generally clustered into three main types based on the shipping addresses:

123 Country Lane, Rural Town - This address often orders garden and
outdoor-related items like soil mix, vegetables, flower seeds, lawn
fertilizer, an outdoor grill, a watering can, and an herb garden kit.

456 City Apt, Metroville - This address tends to order various types of
books, indicating an interest in reading across genres like fiction,
thrillers, biographies, science fiction, literary classics, non-fiction
essays, and children's books.

789 Suburb St, Quietville - The purchases for this address are consistently
baby-related products, such as diapers, wipes, baby formula, infant toys,
baby clothes, and baby bath items.

```

```
Given these patterns, a digital thermometer could potentially be a fit for
any of the addresses based on different needs: outdoor temperature tracking,
a tool for a passionate reader interested in science gadgets, or a health
tool for a baby. However, considering its common use in monitoring health,
especially for babies, it's most likely to be purchased by someone who's
already buying baby care products.

Therefore, the most likely shipping address for a digital thermometer, based
on the purchasing patterns, would be 789 Suburb St, Quietville.
```

Ako mislite da je preterivanje koristiti skup model kao što je GPT-4 za ovaj zadatak, u pravu ste! Isprobao sam isti upit na Mistral 7B Instruct modelu i on je dao sledeći odgovor brzinom od 75 tokena u sekundi, po zanemarljivoj ceni od 0,000218 američkih dolara.

```
Based on the given order history, it appears that the customer has received
orders for baby-related items (diapers, wipes, formula, clothes, bath items)
at the address "789 Suburb St, Quietville" and garden-related items (soil,
seeds, watering can, herb garden kit) at the address "123 Country Lane, Rural
Town". The customer has also received books (fiction, non-fiction, literary
classics, mystery-thriller, science fiction, biographies, children's books)
at the address "456 City Apt, Metroville".

Given that the new order is for a "thermometer", which doesn't seem to fit
into any of the categories that the customer has previously ordered, it's
hard to accurately predict the shipping address based solely on the order
history. However, given the 50%-50% split between baby-related and
garden-related items, it could somewhat lean towards the Baby-related items
address ("789 Suburb St, Quietville"). But remember, this is an assumption
and cannot be definitively confirmed without more context or information.
```

Da li su dodatni napor i troškovi ove tehnike vredni toga da iskustvo kupovine učine magičnijim? Za mnoge online prodavce, apsolutno. A sudeći po svemu, troškovi AI računanja će samo padati, posebno za provajdere hostinga modela otvorenog koda koji su u trci ka dnu.

Koristite Šablon Prompta i StructuredIO zajedno sa Ograđivanjem Odgovora da optimizujete ovakvu vrstu chat kompletiranja.

Adaptivno Redanje Polja

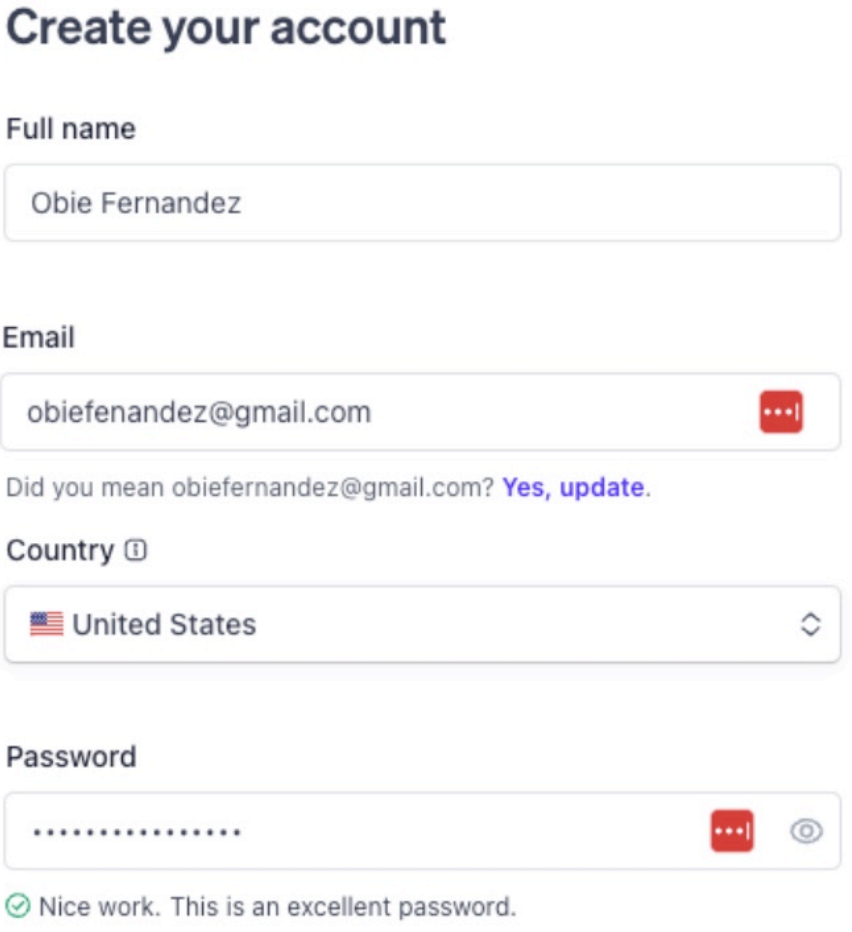
Redosled kojim se prikazuju polja formulara može značajno uticati na korisničko iskustvo i stope završetka. Sa GenUI, možete dinamički prilagoditi redosled polja na osnovu korisničkog konteksta i važnosti svakog polja. Na primer, ako korisnik popunjava registracioni formular za fitness aplikaciju, formular bi mogao dati prioritet poljima vezanim za njihove fitness ciljeve i preference, čineći proces relevantnijim i angažovanijim.

Personalizovani Mikrotekst

Instrukcioni tekst, poruke o greškama i drugi mikrotekst povezan sa formularima takođe se može personalizovati korišćenjem GenUI. Umesto prikazivanja generičkih poruka o greškama poput "Nevažeća email adresa," možete generisati korisnije i kontekstualne poruke kao što je "Molimo unesite važeću email adresu kako biste primili potvrdu vaše porudžbine." Ovi personalizovani detalji mogu učiniti iskustvo popunjavanja formulara pristupačnijim i manje frustrirajućim.

Personalizovana Validacija

U skladu sa Personalizovanim Mikrotekstom, mogli biste koristiti AI za validaciju formulara na načine koji deluju magično. Zamislite da pustite AI da validira formular korisničkog profila, tražeći potencijalne greške na *semantičkom* nivou.



Slika 9. Možete li uočiti semantičku validaciju koja se dešava?

Progresivno Otkrivanje

GenUI može inteligentno odrediti koja su polja formulara suštinska na osnovu korisničkog konteksta i postepeno otkrivati dodatna polja po potrebi. Ova tehnika progresivnog otkrivanja pomaže u smanjenju kognitivnog opterećenja i čini proces popunjavanja formulara lakšim za upravljanje. Na primer, ako se korisnik prijavljuje

za osnovnu pretplatu, formular može inicijalno prikazati samo suštinska polja, a kako korisnik napreduje ili bira određene opcije, dodatna relevantna polja se mogu dinamički uvoditi.

Kontekstualno Svestan Tekst za Objašnjenja

Objašnjenja (tooltip) se često koriste za pružanje dodatnih informacija ili smernica korisnicima kada prelaze mišem preko ili stupaju u interakciju sa određenim elementima. Sa pristupom "Kontekstualnog Generisanja Sadržaja", možete generisati objašnjenja koja se prilagođavaju korisničkom kontekstu i pružaju relevantne informacije. Na primer, ako korisnik istražuje kompleksnu funkciju, objašnjenje može ponuditi personalizovane savete ili primere zasnovane na njihovim prethodnim interakcijama ili nivou veštine.

Tekst za objašnjenja, kao što su instrukcije, opisi ili poruke za pomoć, može se dinamički generisati na osnovu korisničkog konteksta. Umesto predstavljanja generičkih objašnjenja, možete koristiti LLM-ove za generisanje teksta koji je prilagođen specifičnim potrebama ili pitanjima korisnika. Na primer, ako korisnik ima poteškoća sa određenim korakom u procesu, tekst za objašnjenje može pružiti personalizovane smernice ili savete za rešavanje problema.

Mikrotekst se odnosi na male delove teksta koji vode korisnike kroz vašu aplikaciju, kao što su oznake dugmadi, poruke o greškama ili upiti za potvrdu. Primenom pristupa Kontekstualnog Generisanja Sadržaja na mikrotekst, možete kreirati adaptivni UI koji reaguje na korisničke akcije i pruža relevantan i koristan tekst. Na primer, ako korisnik treba da izvrši kritičnu akciju, upit za potvrdu može biti dinamički generisan kako bi pružio jasnu i personalizovanu poruku.

Personalizovani tekst za objašnjenja i objašnjenja mogu značajno poboljšati proces uvođenja novih korisnika. Pružanjem kontekstualno specifičnih smernica i primera, možete pomoći korisnicima da brzo razumeju i navigiraju aplikaciju, smanjujući krivu učenja i povećavajući usvajanje.

Dinamički i kontekstualno svesni chrome elementi takođe mogu učiniti aplikaciju intuitivnijom i angažovanijom. Korisnici će verovatnije stupati u interakciju sa funkcijama i istraživati ih kada je prateći tekst prilagođen njihovim specifičnim potrebama i interesovanjima.

Do sada smo obradili ideje za unapređenje postojećih UI paradigmi pomoću veštačke inteligencije, ali šta je sa preispitivanjem načina na koji se korisnički interfejsi dizajniraju i implementiraju na radikalniji način?

Definisanje generativnog UI-ja

Za razliku od tradicionalnog UI dizajna, gde dizajneri kreiraju fiksne, statične interfejse, GenUI nagoveštava budućnost u kojoj naš softver poseduje fleksibilna, personalizovana iskustva koja mogu da se razvijaju i prilagođavaju u realnom vremenu. Svaki put kada koristimo interfejs za konverzaciju zasnovan na veštačkoj inteligenciji, dozvoljavamo AI-ju da se prilagodi specifičnim potrebama korisnika. GenUI ide korak dalje primenjujući taj nivo prilagodljivosti na *vizuelni* interfejs softvera.

Razlog zbog kojeg je danas moguće eksperimentisati sa GenUI idejama je taj što veliki jezički modeli već razumeju programiranje i njihovo osnovno znanje uključuje UI tehnologije i okvire. Pitanje je, dakle, da li se veliki jezički modeli mogu koristiti za generisanje UI elemenata, kao što su tekst, slike, rasporedi, pa čak i celokupni interfejsi, koji su prilagođeni svakom pojedinačnom korisniku. Model bi mogao biti podešen da uzme u obzir različite faktore, kao što su prethodne interakcije korisnika, navedene preference, demografske informacije i trenutni kontekst upotrebe, kako bi kreirao visoko personalizovane i relevantne interfejse.

GenUI se razlikuje od tradicionalnog dizajna korisničkog interfejsa na nekoliko ključnih načina:

1. **Dinamičan i adaptivan**: Tradicionalni UI dizajn podrazumeva kreiranje fiksnih, statičnih interfejsa koji ostaju isti za sve korisnike. Nasuprot tome, GenUI omogućava interfejse koji se mogu dinamički prilagođavati i menjati na osnovu potreba korisnika i konteksta. To znači da ista aplikacija može predstaviti različite interfejse različitim korisnicima, ili čak istom korisniku u različitim situacijama.
2. **Personalizacija na velikoj skali**: Kod tradicionalnog dizajna, kreiranje personalizovanih iskustava za svakog korisnika često je nepraktično zbog potrebnog vremena i resursa. GenUI, s druge strane, omogućava personalizaciju na velikoj skali. Korišćenjem AI-ja, dizajneri mogu kreirati interfejse koji se automatski prilagođavaju jedinstvenim potrebama i preferencama svakog korisnika, bez potrebe za ručnim dizajniranjem i razvijanjem posebnih interfejsa za svaki segment korisnika.
3. **Fokus na rezultatima**: Tradicionalni UI dizajn često se fokusira na kreiranje vizuelno privlačnih i funkcionalnih interfejsa. Iako su ovi aspekti i dalje važni u GenUI-ju, primarni fokus se pomera ka postizanju željenih korisničkih rezultata. GenUI teži kreiranju interfejsa koji su optimizovani za specifične ciljeve i zadatke svakog korisnika, dajući prioritet upotrebljivosti i efektivnosti u odnosu na čisto estetske aspekte.
4. **Kontinuirano učenje i poboljšanje**: GenUI sistemi mogu kontinuirano da uče i poboljšavaju se tokom vremena na osnovu korisničkih interakcija i povratnih informacija. Dok korisnici koriste generisane interfejse, AI modeli mogu prikupljati podatke o ponašanju korisnika, preferencama i rezultatima, koristeći ove informacije za usavršavanje i optimizaciju budućih generacija interfejsa. Ovaj iterativni proces učenja omogućava GenUI sistemima da vremenom postaju sve efikasniji u zadovoljavanju potreba korisnika.

Važno je napomenuti da GenUI nije isto što i AI-potpomognuti alati za dizajn, poput onih koji pružaju sugestije ili automatizuju određene dizajnerske zadatke. Iako ovi alati mogu biti korisni u pojednostavljivanju procesa dizajna, oni se i dalje oslanjaju na dizajnere da donose konačne odluke i kreiraju statične interfejse. GenUI, s druge

strane, podrazumeva da AI sistem preuzima aktivniju ulogu u stvarnom generisanju i prilagođavanju interfejsa na osnovu korisničkih podataka i konteksta.

GenUI predstavlja značajan pomak u načinu na koji pristupamo dizajnu korisničkog interfejsa, udaljavajući se od univerzalnih rešenja i krećući se ka visoko personalizovanim, adaptivnim iskustvima. Korišćenjem moći veštačke inteligencije, GenUI ima potencijal da revolucionarizuje način na koji komuniciramo sa digitalnim proizvodima i uslugama, stvarajući interfejse koji su intuitivniji, angažovaniji i efikasniji za svakog pojedinačnog korisnika.

Primer

Da bismo ilustrovali koncept GenUI-ja, razmotrimo hipotetičku fitness aplikaciju pod nazivom "FitAI". Ova aplikacija ima za cilj da pruži personalizovane planove vežbanja i savete o ishrani korisnicima na osnovu njihovih individualnih ciljeva, nivoa kondicije i preferencija.

U tradicionalnom pristupu UI dizajnu, FitAI bi mogao imati fiksan set ekrana i elemenata koji su isti za sve korisnike. Međutim, sa GenUI-jem, interfejs aplikacije bi se mogao dinamički prilagođavati jedinstvenim potrebama i kontekstu svakog korisnika.

Ovaj pristup je pomalo teško zamisliti za implementaciju u 2024. godini i možda čak nema adekvatan ROI, ali je moguć.

Evo kako bi to moglo da funkcioniše:

1. **Uvođenje korisnika**:
    - Umesto standardnog upitnika, FitAI koristi konverzacijsku veštačku inteligenciju za prikupljanje informacija o ciljevima korisnika, trenutnom nivou kondicije i preferencama.

- Na osnovu ove početne interakcije, AI generiše personalizovani raspored kontrolne table, naglašavajući funkcije i informacije koje su najrelevantnije za ciljeve korisnika.
- Trenutna AI tehnologija bi mogla imati na raspolaganju izbor komponenti ekrana koje bi koristila u sastavljanju personalizovane kontrolne table.
- Buduća AI tehnologija bi mogla preuzeti ulogu iskusnog UI dizajnera i zapravo kreirati kontrolnu tablu *od nule*.

2. **Planer treninga**:

   - AI prilagođava interfejs planera treninga specifično prema nivou iskustva korisnika i dostupnoj opremi.
   - Za početnika bez opreme, može prikazivati jednostavne vežbe sa sopstvenom težinom uz detaljna uputstva i video zapise.
   - Za naprednog korisnika sa pristupom teretani, može prikazivati složenije rutine sa manje objašnjenja.
   - Sadržaj planera treninga se ne filtrira jednostavno iz velikog skupa podataka. Može se generisati *u realnom vremenu* na osnovu baze znanja koja se pretražuje sa kontekstom koji uključuje sve što se zna o korisniku.

3. **Praćenje napretka**:

   - Interfejs za praćenje napretka razvija se na osnovu korisnikovih ciljeva i obrazaca angažovanja.
   - Ako je korisnik prvenstveno fokusiran na gubitak težine, interfejs može istaknuto prikazivati grafikon trenda težine i statistiku potrošnje kalorija.
   - Za korisnika koji gradi mišićnu masu, mogao bi isticati napredak u snazi i promene u telesnoj kompoziciji.
   - AI može prilagoditi ovaj deo aplikacije stvarnom napretku korisnika. Ako napredak stane na određeni period, aplikacija može preći u režim u kojem pokušava navesti korisnika da otkrije razloge zastoja, kako bi ih ublažila.

4. **Nutricionistički saveti**:
    - Sekcija za ishranu prilagođava se korisnikovim prehrambenim preferencijama i ograničenjima.
    - Za vegana, može prikazivati predloge biljnih obroka i izvore proteina.
    - Za korisnika sa netolerancijom na gluten, automatski bi filtrirala namirnice koje sadrže gluten iz preporuka.
    - Ponovo, sadržaj se ne izvlači iz ogromnog skupa podataka o obrocima koji važi za sve korisnike, već se sintetizuje iz baze znanja koja sadrži informacije prilagodljive specifičnoj situaciji i ograničenjima korisnika.
    - Na primer, recepti se generišu sa specifikacijama sastojaka koje odgovaraju konstantno promenjivim kalorijskim potrebama korisnika kako se njihov nivo kondicije i telesne statistike razvijaju.
5. **Motivacioni elementi**:
    - Motivacioni sadržaj i obaveštenja aplikacije personalizovani su na osnovu tipa ličnosti korisnika i reakcije na različite motivacione strategije.
    - Neki korisnici mogu primati ohrabrujuće poruke, dok drugi dobijaju povratne informacije više zasnovane na podacima.

U ovom primeru, GenUI omogućava FitAI-u da stvori visoko prilagođeno iskustvo za svakog korisnika, potencijalno povećavajući angažovanje, zadovoljstvo i verovatnoću postizanja fitnes ciljeva. Elementi interfejsa, sadržaj, pa čak i "ličnost" aplikacije prilagođavaju se kako bi najbolje služili potrebama i preferencijama svakog pojedinačnog korisnika.

Prelazak na dizajn orijentisan ka ishodima

GenUI predstavlja fundamentalni pomak u pristupu dizajnu korisničkog interfejsa, prelazeći sa fokusa na kreiranje specifičnih elemenata interfejsa na više holistički pristup orijentisan ka ishodima. Ovaj pomak ima nekoliko važnih implikacija:

1. **Fokus na ciljeve korisnika**:

    - Dizajneri će morati dublje razmišljati o ciljevima korisnika i željenim ishodima umesto o specifičnim komponentama interfejsa.
    - Naglasak će biti na stvaranju sistema koji mogu generisati interfejse koji pomažu korisnicima da efikasno i efektivno postignu svoje ciljeve.
    - Pojaviće se novi UI okviri koji će AI-zasnovanim dizajnerima dati alate potrebne za generisanje korisničkih iskustava *u realnom vremenu* i *iz početka* umesto na osnovu unapred definisanih specifikacija ekrana.

2. **Promena uloge dizajnera**:

    - Dizajneri će preći sa kreiranja fiksnih izgleda na definisanje pravila, ograničenja i smernica koje AI sistemi treba da prate pri generisanju interfejsa.
    - Moraće da razviju veštine u oblastima kao što su analiza podataka, inženjering AI upita i sistemsko razmišljanje kako bi efikasno vodili GenUI sisteme.

3. **Važnost istraživanja korisnika**:

    - Istraživanje korisnika postaje još kritičnije u GenUI kontekstu, jer dizajneri moraju razumeti ne samo preferencije korisnika, već i kako se te preferencije i potrebe menjaju u različitim kontekstima.
    - Kontinuirano testiranje korisnika i povratne informacije biće ključni za usavršavanje i poboljšanje sposobnosti AI-ja da generiše efikasne interfejse.

4. **Dizajniranje za varijabilnost**:

    - Umesto kreiranja jednog "savršenog" interfejsa, dizajneri će morati da razmotre više mogućih varijacija i osiguraju da sistem može generisati odgovarajuće interfejse za različite potrebe korisnika.

- Ovo uključuje dizajniranje za granične slučajeve i osiguravanje da generisani interfejsi održavaju upotrebljivost i pristupačnost kroz različite konfiguracije.
- Diferencijacija proizvoda dobija nove dimenzije koje uključuju divergentne perspektive o psihologiji korisnika i korišćenje jedinstvenih skupova podataka i baza znanja nedostupnih konkurentima.

Izazovi i razmatranja

Iako GenUI nudi uzbudljive mogućnosti, takođe predstavlja nekoliko izazova i razmatranja:

1. **Tehnička ograničenja**:
    - Trenutna AI tehnologija, iako napredna, i dalje ima ograničenja u razumevanju složenih namera korisnika i generisanju istinski kontekstualno svesnih interfejsa.
    - Problemi sa performansama vezani za generisanje elemenata interfejsa u realnom vremenu, posebno na manje snažnim uređajima.
2. **Zahtevi za podacima**:
    - U zavisnosti od slučaja upotrebe, efikasni GenUI sistemi mogu zahtevati značajne količine korisničkih podataka za generisanje personalizovanih interfejsa.
    - Izazovi u etičkom prikupljanju autentičnih korisničkih podataka pokreću pitanja o privatnosti i bezbednosti podataka, kao i o potencijalnim pristrasnostima u podacima koji se koriste za obuku GenUI modela.
3. **Upotrebljivost i doslednost**:

- Barem dok praksa ne postane široko rasprostranjena, aplikacija sa konstantno promenljivim interfejsima mogla bi dovesti do problema upotrebljivosti, jer bi korisnici mogli imati poteškoća u pronalaženju poznatih elemenata ili efikasnoj navigaciji.
- Postizanje ravnoteže između personalizacije i održavanja doslednog, savladivog interfejsa biće ključno.

4. **Preterano oslanjanje na AI**:

   - Postoji rizik od preteranog delegiranja odluka o dizajnu AI sistemima, što potencijalno može dovesti do neinspirativnih, problematičnih ili jednostavno neispravnih izbora interfejsa.
   - Ljudski nadzor i mogućnost preglasavanja AI-generisanih dizajna ostaće važni u doglednoj budućnosti.

5. **Pitanja pristupačnosti**:

   - Osiguravanje da dinamički generisani interfejsi ostanu pristupačni korisnicima sa invaliditetom predstavlja potpuno nove izazove, što je zabrinjavajuće s obzirom na loš nivo usklađenosti sa pristupačnošću koji pokazuju tipični sistemi.
   - S druge strane, AI dizajneri mogu biti implementirani sa *ugrađenom* brigom za pristupačnost i mogućnostima za izgradnju pristupačnih interfejsa u hodu, baš kao što grade UI za korisnike bez oštećenja.
   - U svakom slučaju, GenUI sistemi bi trebalo da budu dizajnirani sa robusnim smernicama za pristupačnost i procesima testiranja.

6. **Poverenje korisnika i transparentnost**:

   - Korisnici se mogu osećati nelagodno sa interfejsima koji naizgled "znaju previše" o njima ili se menjaju na načine koje ne razumeju.
   - Obezbeđivanje transparentnosti o tome kako i zašto se interfejsi personalizuju biće važno za izgradnju poverenja korisnika.

Budući izgledi i mogućnosti

Budućnost Generativnog UI-ja (GenUI) nosi ogromno obećanje za revoluciju načina na koji komuniciramo sa digitalnim proizvodima i uslugama. Kako se ova tehnologija nastavlja razvijati, možemo očekivati seizmičku promenu u načinu na koji se korisnički interfejsi dizajniraju, implementiraju i doživljavaju. Mislim da je GenUI fenomen koji će konačno gurnuti naš softver u domen onoga što se sada smatra naučnom fantastikom.

Jedna od najuzbudljivijih perspektiva GenUI-ja je njegov potencijal da unapredi pristupačnost u obimu koji prevazilazi samo osiguravanje da ljudi sa ozbiljnim invaliditetom nisu potpuno isključeni iz korišćenja vašeg softvera. Automatskim prilagođavanjem interfejsa individualnim potrebama korisnika, GenUI bi mogao učiniti digitalna iskustva inkluzivnijim nego ikad pre. Zamislite interfejse koji se besprekorno prilagođavaju da obezbede veći tekst za mlađe ili vizuelno oštećene korisnike ili pojednostavljene rasporede za one sa kognitivnim poteškoćama, sve bez potrebe za ručnom konfiguracijom ili posebnim "pristupačnim" verzijama aplikacija.

Mogućnosti personalizacije GenUI-ja će verovatno dovesti do povećanog angažovanja korisnika, zadovoljstva i lojalnosti kroz širok spektar digitalnih proizvoda. Kako interfejsi postaju više usklađeni sa individualnim preferencijama i ponašanjima, korisnici će smatrati digitalna iskustva intuitivnijim i prijatnijim, što potencijalno vodi do dubljih i smislenijih interakcija sa tehnologijom.

GenUI takođe ima potencijal da transformiše proces uvođenja novih korisnika. Stvaranjem intuitivnih, personalizovanih iskustava za nove korisnike koja se brzo prilagođavaju nivou stručnosti svakog korisnika, GenUI bi mogao značajno smanjiti krivu učenja povezanu sa novim aplikacijama. Ovo bi moglo dovesti do bržih stopa usvajanja i povećanog samopouzdanja korisnika u istraživanju novih funkcija i funkcionalnosti.

Još jedna uzbudljiva mogućnost je sposobnost GenUI-ja da održava konzistentno korisničko iskustvo na različitim uređajima i platformama, dok optimizuje za svaki

specifični kontekst upotrebe. Ovo bi moglo rešiti dugogodišnji izazov pružanja koherentnih iskustava kroz sve fragmentiraniji pejzaž uređaja, od pametnih telefona i tableta do desktop računara i tehnologija u nastajanju poput naočara za proširenu realnost.

Priroda GenUI-ja zasnovana na podacima otvara mogućnosti za brzu iteraciju i poboljšanje u UI dizajnu. Prikupljanjem podataka u realnom vremenu o tome kako korisnici komuniciraju sa generisanim interfejsima, dizajneri i programeri mogu steći neprevaziđene uvide u ponašanje i preferencije korisnika. Ova povratna petlja mogla bi dovesti do kontinuiranih poboljšanja u UI dizajnu, vođenih stvarnim obrascima korišćenja umesto pretpostavkama ili ograničenim korisničkim testiranjem.

Da bi se pripremili za ovu promenu, dizajneri će morati da razviju svoje veštine i način razmišljanja. Fokus će se pomeriti sa kreiranja fiksnih rasporeda na razvoj sveobuhvatnih sistema dizajna i smernica koje mogu informisati generisanje interfejsa vođeno AI-jem. Dizajneri će morati da razviju duboko razumevanje analize podataka, AI tehnologija i sistemskog razmišljanja kako bi efikasno vodili GenUI sisteme.

Štaviše, kako GenUI zamagljuje granice između dizajna i tehnologije, dizajneri će morati bliže da sarađuju sa programerima i naučnicima koji se bave podacima. Ovaj interdisciplinarni pristup biće ključan u stvaranju GenUI sistema koji nisu samo vizuelno privlačni i prijateljski nastrojeni prema korisnicima, već i tehnički robusni i etički ispravni.

Etičke implikacije GenUI-ja će takođe doći u prvi plan kako tehnologija sazreva. Dizajneri će imati ključnu ulogu u razvoju okvira za odgovornu upotrebu veštačke inteligencije u dizajnu interfejsa, osiguravajući da personalizacija unapređuje korisničko iskustvo bez ugrožavanja privatnosti ili neetičke manipulacije ponašanjem korisnika.

Gledajući u budućnost, GenUI predstavlja i uzbudljive mogućnosti i značajne izazove. Ima potencijal da stvori intuitivnija, efikasnija i zadovoljavajuća digitalna iskustva za korisnike širom sveta. Iako će zahtevati od dizajnera da se prilagode i steknu nove veštine, takođe pruža neviđenu priliku da se oblikuje budućnost interakcije između

čoveka i računara na dubok i smislen način. Put ka potpuno realizovanim GenUI sistemima će nesumnjivo biti složen, ali potencijalne nagrade u smislu poboljšanog korisničkog iskustva i digitalne pristupačnosti čine ga budućnošću za koju vredi težiti.

Inteligentna orkestracija radnih tokova

U domenu razvoja aplikacija, radni tokovi igraju ključnu ulogu u definisanju kako se zadaci, procesi i interakcije korisnika strukturiraju i izvršavaju. Kako aplikacije postaju složenije, a očekivanja korisnika nastavljaju da rastu, potreba za inteligentnom i adaptivnom orkestracijom radnih tokova postaje sve očiglednija.

Pristup "Inteligentne orkestracije radnih tokova" fokusira se na korišćenje AI komponenti za dinamičku orkestraciju i optimizaciju složenih radnih tokova unutar aplikacija. Cilj je stvaranje aplikacija koje su efikasnije, responzivnije i prilagodljivije podacima i kontekstu u realnom vremenu.

U ovom poglavlju istražićemo ključne principe i obrasce koji podupiru pristup inteligentne orkestracije radnih tokova. Razmotrićemo kako se AI može koristiti za

inteligentno usmeravanje zadataka, automatizaciju donošenja odluka i dinamičko prilagođavanje radnih tokova na osnovu različitih faktora kao što su ponašanje korisnika, performanse sistema i poslovna pravila. Kroz praktične primere i scenarije iz stvarnog sveta, demonstriraćemo transformativni potencijal AI-ja u pojednostavljivanju i optimizaciji radnih tokova aplikacija.

Bez obzira da li gradite poslovne aplikacije sa složenim poslovnim procesima ili aplikacije namenjene potrošačima sa dinamičkim korisničkim putanjama, obrasci i tehnike o kojima se govori u ovom poglavlju opremić će vas znanjem i alatima za kreiranje inteligentnih i efikasnih radnih tokova koji poboljšavaju celokupno korisničko iskustvo i donose poslovnu vrednost.

Poslovna potreba

Tradicionalni pristupi upravljanju radnim tokovima često se oslanjaju na unapred definisana pravila i statička stabla odlučivanja, koja mogu biti kruta, nefleksibilna i nesposobna da se nose sa dinamičnom prirodom modernih aplikacija.

Razmotrite scenario gde aplikacija za elektronsku trgovinu treba da upravlja složenim procesom ispunjenja porudžbine. Radni tok može uključivati više koraka kao što su validacija porudžbine, provera zaliha, obrada plaćanja, isporuka i obaveštenja kupcima. Svaki korak može imati svoj set pravila, zavisnosti, eksterne integracije i mehanizme za rukovanje izuzecima. Upravljanje takvim radnim tokom ručno ili kroz hardkodirani kod može brzo postati nezgrapno, sklono greškama i teško za održavanje.

Štaviše, kako aplikacija skalira i broj istovremenih korisnika raste, radni tok će možda morati da se prilagođava i optimizuje na osnovu podataka u realnom vremenu i performansi sistema. Na primer, tokom perioda vršnog opterećenja, aplikacija će možda morati dinamički da prilagodi radni tok kako bi prioritizovala određene zadatke, efikasno raspodelila resurse i osigurala nesmetano korisničko iskustvo.

Tu nastupa pristup “Inteligentne orkestracije radnih tokova”. Korišćenjem AI

komponenti, programeri mogu kreirati radne tokove koji su inteligentni, adaptivni i samo-optimizujući. AI može analizirati ogromne količine podataka, učiti iz prošlih iskustava i donositi informisane odluke u realnom vremenu za efikasnu orkestraciju radnog toka.

Ključne prednosti

1. **Povećana efikasnost**: AI može optimizovati raspoređivanje zadataka, korišćenje resursa i izvršavanje radnih tokova, što dovodi do bržeg vremena obrade i poboljšane ukupne efikasnosti.
2. **Prilagodljivost**: Radni tokovi vođeni AI-jem mogu se dinamički prilagođavati promenjivim uslovima, kao što su fluktuacije u korisničkoj potražnji, performansama sistema ili poslovnim zahtevima, osiguravajući da aplikacija ostane responzivna i otporna.
3. **Automatizovano donošenje odluka**: AI može automatizovati složene procese donošenja odluka unutar radnog toka, smanjujući manuelne intervencije i minimizujući rizik od ljudskih grešaka.
4. **Personalizacija**: AI može analizirati ponašanje korisnika, preference i kontekst kako bi personalizovao radni tok i isporučio prilagođena iskustva pojedinačnim korisnicima.
5. **Skalabilnost**: Radni tokovi pokretani AI-jem mogu se besprekorno skalirati kako bi rukovali rastućim obimom podataka i korisničkih interakcija, bez ugrožavanja performansi ili pouzdanosti.

U sledećim odeljcima istražićemo ključne obrasce i tehnike koje omogućavaju implementaciju inteligentnih radnih tokova i prikazati primere iz stvarnog sveta o tome kako AI transformiše upravljanje radnim tokovima u modernim aplikacijama.

Ključni obrasci

Za implementaciju inteligentne orkestracije radnih tokova u aplikacijama, programeri mogu iskoristiti nekoliko ključnih obrazaca koji koriste snagu AI-ja. Ovi obrasci pružaju strukturirani pristup dizajniranju i upravljanju radnim tokovima, omogućavajući aplikacijama da se prilagođavaju, optimizuju i automatizuju procese na osnovu podataka i konteksta u realnom vremenu. Hajde da istražimo neke od fundamentalnih obrazaca u inteligentnoj orkestraciji radnih tokova.

Dinamičko usmeravanje zadataka

Ovaj obrazac uključuje korišćenje AI-ja za inteligentno usmeravanje zadataka unutar radnog toka na osnovu različitih faktora kao što su prioritet zadatka, dostupnost resursa i performanse sistema. AI algoritmi mogu analizirati karakteristike svakog zadatka, uzeti u obzir trenutno stanje sistema i donositi informisane odluke za dodeljivanje zadataka najprikladnijim resursima ili putanjama obrade. Dinamičko usmeravanje zadataka osigurava da su zadaci efikasno distribuirani i izvršeni, optimizujući ukupne performanse radnog toka.

```
class TaskRouter
  include Raix::ChatCompletion
  include Raix::FunctionDispatch

  attr_accessor :task

  # list of functions that can be called by the AI entirely at its
  # discretion depending on the task received

  function :analyze_task_priority do
    TaskPriorityAnalyzer.perform(task)
  end

  function :check_resource_availability, # ...
  function :assess_system_performance, # ...
```

```
16    function :assign_task_to_resource, # ...
17
18    DIRECTIVE = "You are a task router, responsible for intelligently
19     assigning tasks to available resources based on priority, resource
20     availability, and system performance..."
21
22    def initialize(task)
23      self.task = task
24      transcript << { system: DIRECTIVE }
25      transcript << { user: task.to_json }
26    end
27
28    def perform
29      while task.unassigned?
30        chat_completion
31
32        # todo: add max loop counter and break
33      end
34
35      # capture the transcript for later analysis
36      task.update(routing_transcript: transcript)
37    end
38  end
```

Obratite pažnju na petlju kreiranu while izrazom u liniji 29, koja nastavlja da šalje upite AI-ju sve dok zadatak nije dodeljen. U liniji 35, čuvamo transkript zadatka za kasniju analizu i otklanjanje grešaka, ako bude potrebno.

Kontekstualno Donošenje Odluka

Možete koristiti vrlo sličan kod za donošenje odluka zasnovanih na kontekstu unutar toka rada. Analiziranjem relevantnih podataka kao što su korisničke preference, istorijski obrasci i unosi u realnom vremenu, AI komponente mogu odrediti najprikladniji tok akcije na svakoj tački odlučivanja u toku rada. Prilagodite ponašanje vašeg toka rada na osnovu specifičnog konteksta svakog korisnika ili scenarija, pružajući personalizovana i optimizovana iskustva.

Adaptivna Kompozicija Toka Rada

Ovaj obrazac se fokusira na dinamičko komponovanje i prilagođavanje tokova rada na osnovu promenljivih zahteva ili uslova. AI može analizirati trenutno stanje toka rada, identifikovati uska grla ili neefikasnosti, i automatski modifikovati strukturu toka rada kako bi optimizovao performanse. Adaptivna kompozicija toka rada omogućava aplikacijama da se kontinuirano razvijaju i poboljšavaju svoje procese bez potrebe za ručnom intervencijom.

Rukovanje Izuzecima i Oporavak

Rukovanje izuzecima i oporavak su kritični aspekti inteligentne orkestracije toka rada. Kada radite sa AI komponentama i složenim tokovima rada, neophodno je predvideti i elegantno rukovati izuzecima kako bi se osigurala stabilnost i pouzdanost sistema.

Evo nekoliko ključnih razmatranja i tehnika za rukovanje izuzecima i oporavak u inteligentnim tokovima rada:

1. **Propagacija Izuzetaka**: Implementirajte konzistentan pristup za propagaciju izuzetaka kroz komponente toka rada. Kada se izuzetak pojavi unutar komponente, treba ga uhvatiti, zabeležiti i propagirati do orkestratora ili zasebne komponente odgovorne za rukovanje izuzecima. Ideja je da se centralizuje rukovanje izuzecima i spreči tiho gutanje izuzetaka, kao i otvaranje mogućnosti za Inteligentno Rukovanje Greškama.
2. **Mehanizmi Ponovnog Pokušaja**: Mehanizmi ponovnog pokušaja pomažu u poboljšanju otpornosti toka rada i elegantnom rukovanju povremenim greškama. Svakako pokušajte implementirati mehanizme ponovnog pokušaja za prolazne ili oporavljive izuzetke, kao što su mrežna povezanost ili nedostupnost resursa koji se mogu automatski ponovno pokušati nakon određenog kašnjenja. Posedovanje AI-pogonjenog orkestratora ili rukovaoca izuzecima znači da vaše strategije

ponovnog pokušaja ne moraju biti mehaničke prirode, oslanjajući se na fiksne algoritme poput eksponencijalnog povlačenja. Možete prepustiti rukovanje ponovnim pokušajem "diskreciji" AI komponente odgovorne za odlučivanje kako rukovati izuzetkom.

3. **Rezervne Strategije**: Ako AI komponenta ne uspe da pruži validan odgovor ili naiđe na grešku--što je česta pojava s obzirom na njenu naprednu prirodu--imajte rezervni mehanizam koji će osigurati da tok rada može da se nastavi. Ovo može uključivati korišćenje podrazumevanih vrednosti, alternativnih algoritama, ili Čoveka u Petlji za donošenje odluka i održavanje napretka toka rada.
4. **Kompenzacione Akcije**: Direktive orkestratora treba da uključuju uputstva o kompenzacionim akcijama za rukovanje izuzecima koji se ne mogu automatski rešiti. Kompenzacione akcije su koraci koji se preduzimaju da bi se poništili ili ublažili efekti neuspele operacije. Na primer, ako korak obrade plaćanja ne uspe, kompenzaciona akcija bi mogla biti vraćanje transakcije i obaveštavanje korisnika. Kompenzacione akcije pomažu u održavanju konzistentnosti podataka i integriteta u slučaju izuzetaka.
5. **Praćenje i Upozoravanje o Izuzecima**: Postavite mehanizme za praćenje i upozoravanje kako biste otkrili i obavestili relevantne zainteresovane strane o kritičnim izuzecima. Orkestrator može biti svestan pragova i pravila za pokretanje upozorenja kada izuzeci pređu određene granice ili kada se pojave specifični tipovi izuzetaka. Ovo omogućava proaktivnu identifikaciju i rešavanje problema pre nego što utiču na celokupni sistem.

Evo primera rukovanja izuzecima i oporavka u Ruby komponenti toka rada:

```
class InventoryManager
  def check_availability(order)
    begin
      # Perform inventory check logic
      inventory = Inventory.find_by(product_id: order.product_id)
      if inventory.available_quantity >= order.quantity
        return true
      else
        raise InsufficientInventoryError,
              "Insufficient inventory for product #{order.product_id}"
      end
    rescue InsufficientInventoryError => e
      # Log the exception
      logger.error("Inventory check failed: #{e.message}")

      # Retry the operation after a delay
      retry_count ||= 0
      if retry_count < MAX_RETRIES
        retry_count += 1
        sleep(RETRY_DELAY)
        retry
      else
        # Fallback to manual intervention
        NotificationService.admin("Inventory check failed: Order #{order.id}")
        return false
      end
    end
  end
end
```

U ovom primeru, InventoryManager komponenta proverava dostupnost proizvoda za datu narudžbinu. Ako je dostupna količina nedovoljna, podiže se InsufficientInventoryError. Izuzetak se hvata, beleži, i implementira se mehanizam ponovnog pokušaja. Ako se prekorači limit ponovnih pokušaja, komponenta prelazi na ručnu intervenciju obaveštavanjem administratora.

Implementacijom robusnog rukovanja izuzecima i mehanizama oporavka, možete osigurati da vaši inteligentni radni tokovi budu otporni, održivi i sposobni da elegantno

rukuju neočekivanim situacijama.

Ovi obrasci čine osnovu orkestracije inteligentnog radnog toka i mogu se kombinovati i prilagoditi specifičnim zahtevima različitih aplikacija. Koristeći ove obrasce, programeri mogu kreirati radne tokove koji su fleksibilni, otporni i optimizovani za performanse i korisničko iskustvo.

U sledećem odeljku, istražićemo kako se ovi obrasci mogu implementirati u praksi, koristeći primere iz stvarnog sveta i isečke koda da bismo ilustrovali integraciju AI komponenti u upravljanje radnim tokovima.

Implementacija orkestracije inteligentnog radnog toka u praksi

Sada kada smo istražili ključne obrasce u orkestraciji inteligentnog radnog toka, hajde da se udubimo u to kako se ovi obrasci mogu implementirati u aplikacijama iz stvarnog sveta. Pružićemo praktične primere i isečke koda da bismo ilustrovali integraciju AI komponenti u upravljanje radnim tokovima.

Inteligentni procesor narudžbina

Hajde da se udubimo u praktičan primer implementacije orkestracije inteligentnog radnog toka koristeći AI-podržanu OrderProcessor komponentu u Ruby on Rails e-commerce aplikaciji. OrderProcessor realizuje koncept Process Manager Enterprise Integration koji smo prvi put sreli u Poglavlju 3 kada smo diskutovali o Mnoštvu radnika. Komponenta će biti odgovorna za upravljanje tokom ispunjavanja narudžbina, donošenje odluka o rutiranju na osnovu međurezultata i orkestraciju izvršavanja različitih koraka obrade.

Proces ispunjavanja narudžbine uključuje više koraka kao što su validacija narudžbine, provera inventara, obrada plaćanja i isporuka. Svaki korak je implementiran kao zaseban radni proces koji obavlja specifičan zadatak i vraća rezultat OrderProcessor-u. Koraci nisu obavezni i ne moraju čak ni da se izvršavaju određenim redosledom.

Evo primera implementacije OrderProcessor-a. Sadrži dva mixin-a iz Raix. Prvi (ChatCompletion) daje mu mogućnost chat completion-a, što ga čini AI komponentom. Drugi (FunctionDispatch) omogućava pozivanje funkcija od strane AI-a, dozvoljavajući mu da odgovori na prompt pozivom funkcije umesto tekstualne poruke.

Radne funkcije (validate_order, check_inventory, i ostale) delegiraju svojim odgovarajućim radnim klasama, koje mogu biti AI ili ne-AI komponente, sa jedinim zahtevom da vrate rezultate svog rada u formatu koji se može predstaviti kao string.

Kao i sa svim drugim primerima u ovom delu knjige, ovaj kod je praktično pseudo-kod i namenjen je samo da prenese značenje obrasca i inspiriše vaše sopstvene kreacije. Potpuni opisi obrazaca i kompletni primeri koda su uključeni u Delu 2.

```
class OrderProcessor
  include Raix::ChatCompletion
  include Raix::FunctionDispatch

  SYSTEM_DIRECTIVE = "You are an order processor, tasked with..."

  def initialize(order)
    self.order = order
    transcript << { system: SYSTEM_DIRECTIVE }
    transcript << { user: order.to_json }
  end

  def perform
    # will continue looping until `stop_looping!` is called
    chat_completion(loop: true)
  end

```

```
18    # list of functions available to be called by the AI
19    # truncated for brevity
20
21    def functions
22      [
23        {
24          name: "validate_order",
25          description: "Invoke to check validity of order",
26          parameters: {
27            ...
28        },
29        ...
30      ]
31    end
32
33    # implementation of functions that can be called by the AI
34    # entirely at its discretion, depending on the needs of the order
35
36    def validate_order
37      OrderValidationWorker.perform(@order)
38    end
39
40    def check_inventory
41      InventoryCheckWorker.perform(@order)
42    end
43
44    def process_payment
45      PaymentProcessingWorker.perform(@order)
46    end
47
48    def schedule_shipping
49      ShippingSchedulerWorker.perform(@order)
50    end
51
52    def send_confirmation
53      OrderConfirmationWorker.perform(@order)
54    end
55
56    def finished_processing
57      @order.update!(transcript:, processed_at: Time.current)
58      stop_looping!
59    end
```

```
60 end
```

U primeru, OrderProcessor se inicijalizuje sa objektom porudžbine i održava transkript izvršavanja toka rada, u tipičnom formatu transkripta konverzacije koji je svojstven velikim jezičkim modelima. Veštačkoj inteligenciji je data potpuna kontrola nad orkestracijom izvršavanja različitih koraka obrade, kao što su validacija porudžbine, provera zaliha, obrada plaćanja i isporuka.

Svaki put kada se pozove metoda chat_completion, transkript se šalje veštačkoj inteligenciji da bi ona obezbedila završetak u vidu poziva funkcije. U potpunosti je na veštačkoj inteligenciji da analizira rezultat prethodnog koraka i odredi odgovarajuću akciju koju treba preduzeti. Na primer, ako provera zaliha otkrije nizak nivo zaliha, OrderProcessor može da zakaže zadatak dopune. Ako obrada plaćanja ne uspe, može da pokrene ponovni pokušaj ili da obavesti korisničku podršku.

> Gornji primer nema definisane funkcije za dopunu zaliha ili obaveštavanje korisničke podrške, ali apsolutno bi mogao da ih ima.

Transkript raste svaki put kada se pozove funkcija i služi kao evidencija izvršavanja toka rada, uključujući rezultate svakog koraka i uputstva koja generiše veštačka inteligencija za sledeće korake. Ovaj transkript se može koristiti za otklanjanje grešaka, reviziju i obezbeđivanje vidljivosti u procesu ispunjenja porudžbine.

Korišćenjem veštačke inteligencije u OrderProcessor-u, e-commerce aplikacija može dinamički da prilagodi tok rada na osnovu podataka u realnom vremenu i inteligentno upravlja izuzecima. Komponenta veštačke inteligencije može da donosi informisane odluke, optimizuje tok rada i osigura neometanu obradu porudžbina čak i u složenim scenarijima.

Činjenica da je jedini zahtev za radne procese da vrate neki razumljiv izlaz koji će

veštačka inteligencija razmotriti pri odlučivanju šta dalje da radi, možda će vam pomoći da shvatite kako ovaj pristup može da smanji posao mapiranja ulaza/izlaza koji je obično potreban pri integraciji različitih sistema međusobno.

Inteligentni moderator sadržaja

Aplikacije društvenih mreža generalno zahtevaju bar minimalnu moderaciju sadržaja kako bi se osigurala bezbedna i zdrava zajednica. Ovaj primer komponente ContentModerator koristi veštačku inteligenciju za inteligentno orkestriranje toka moderacije, donoseći odluke na osnovu karakteristika sadržaja i rezultata različitih koraka moderacije.

Proces moderacije uključuje više koraka kao što su analiza teksta, prepoznavanje slika, procena reputacije korisnika i ručni pregled. Svaki korak je implementiran kao zaseban radni proces koji obavlja određeni zadatak i vraća rezultat ContentModerator-u.

Evo primera implementacije ContentModerator-a:

```
class ContentModerator
  include Raix::ChatCompletion
  include Raix::FunctionDispatch

  SYSTEM_DIRECTIVE = "You are a content moderator process manager,
    tasked with the workflow involved in moderating user-generated content..."

  def initialize(content)
    @content = content
    @transcript = [
      { system: SYSTEM_DIRECTIVE },
      { user: content.to_json }
    ]
  end

  def perform
    complete(@transcript)
  end

```

```
20    def model
21      "openai/gpt-4"
22    end
23
24    # list of functions available to be called by the AI
25    # truncated for brevity
26
27    def functions
28      [
29        {
30          name: "analyze_text",
31          # ...
32        },
33        {
34          name: "recognize_image",
35          description: "Invoke to describe images...",
36          # ...
37        },
38        {
39          name: "assess_user_reputation",
40          # ...
41        },
42        {
43          name: "escalate_to_manual_review",
44          # ...
45        },
46        {
47          name: "approve_content",
48          # ...
49        },
50        {
51          name: "reject_content",
52          # ...
53        }
54      ]
55    end
56
57    # implementation of functions that can be called by the AI
58    # entirely at its discretion, depending on the needs of the order
59
60    def analyze_text
61      result = TextAnalysisWorker.perform(@content)
```

```
    continue_with(result)
  end

  def recognize_image
    result = ImageRecognitionWorker.perform(@content)
    continue_with(result)
  end

  def assess_user_reputation
    result = UserReputationWorker.perform(@content.user)
    continue_with(result)
  end

  def escalate_to_manual_review
    ManualReviewWorker.perform(@content)
    @content.update!(status: 'pending', transcript: @transcript)
  end

  def approve_content
    @content.update!(status: 'approved', transcript: @transcript)
  end

  def reject_content
    @content.update!(status: 'rejected', transcript: @transcript)
  end

  private

  def continue_with(result)
    @transcript << { function: result }
    complete(@transcript)
  end
end
```

U ovom primeru, ContentModerator je inicijalizovan sa objektom sadržaja i održava zapisnik moderacije u formatu konverzacije. AI komponenta ima potpunu kontrolu nad tokom moderacije, odlučujući koje korake da izvrši na osnovu karakteristika sadržaja i rezultata svakog koraka.

Dostupne radne funkcije koje AI može da pozove uključuju analyze_text,

recognize_image, assess_user_reputation i escalate_to_manual_review. Svaka funkcija delegira zadatak odgovarajućem radnom procesu (TextAnalysisWorker, ImageRecognitionWorker, itd.) i dodaje rezultat u zapisnik moderacije, sa izuzetkom funkcije za eskalaciju koja deluje kao završno stanje. Konačno, funkcije approve_content i reject_content takođe deluju kao završna stanja.

AI komponenta analizira sadržaj i određuje odgovarajuću akciju. Ako sadržaj sadrži reference na slike, može pozvati radni proces recognize_image za pomoć pri vizuelnom pregledu. Ako bilo koji radni proces upozori na potencijalno štetan sadržaj, AI može odlučiti da eskalira sadržaj na manuelni pregled ili ga jednostavno odmah odbiti. Ali u zavisnosti od ozbiljnosti upozorenja, AI može odlučiti da koristi rezultate procene reputacije korisnika pri odlučivanju kako da postupi sa sadržajem u kom nije siguran. U zavisnosti od slučaja upotrebe, možda pouzdani korisnici imaju više slobode u onome što mogu da objave. I tako dalje…

Kao i u prethodnom primeru upravitelja procesa, zapisnik moderacije služi kao evidencija izvršavanja toka rada, uključujući rezultate svakog koraka i odluke koje je generisao AI. Ovaj zapisnik se može koristiti za reviziju, transparentnost i poboljšanje procesa moderacije tokom vremena.

Korišćenjem AI-ja u ContentModerator-u, aplikacija društvenih medija može dinamički prilagoditi tok moderacije na osnovu karakteristika sadržaja i inteligentno upravljati složenim scenarijima moderacije. AI komponenta može donositi informisane odluke, optimizovati tok rada i osigurati bezbedno i zdravo iskustvo zajednice.

Hajde da istražimo još dva primera koji pokazuju prediktivno raspoređivanje zadataka i rukovanje izuzecima i oporavak u kontekstu inteligentnog orkestiranja toka rada.

Prediktivno raspoređivanje zadataka u sistemu korisničke podrške

U aplikaciji za korisničku podršku izrađenoj pomoću Ruby on Rails, efikasno upravljanje i prioritizacija tiketa za podršku su ključni za pružanje pravovremene

pomoći korisnicima. Komponenta SupportTicketScheduler koristi AI za prediktivno raspoređivanje i dodeljivanje tiketa za podršku dostupnim agentima na osnovu različitih faktora kao što su hitnost tiketa, stručnost agenta i radno opterećenje.

```
class SupportTicketScheduler
  include Raix::ChatCompletion
  include Raix::FunctionDispatch

  SYSTEM_DIRECTIVE = "You are a support ticket scheduler,
    tasked with intelligently assigning tickets to available agents..."

  def initialize(ticket)
    @ticket = ticket
    @transcript = [
      { system: SYSTEM_DIRECTIVE },
      { user: ticket.to_json }
    ]
  end

  def perform
    complete(@transcript)
  end

  def model
    "openai/gpt-4"
  end

  def functions
    [
      {
        name: "analyze_ticket_urgency",
        # ...
      },
      {
        name: "list_available_agents",
        description: "Includes expertise of available agents",
        # ...
      },
      {
        name: "predict_agent_workload",
        description: "Uses historical data to predict upcoming workloads",
```

```
        # ...
      },
      {
        name: "assign_ticket_to_agent",
        # ...
      },
      {
        name: "reschedule_ticket",
        # ...
      }
    ]
  end

  # implementation of functions that can be called by the AI
  # entirely at its discretion, depending on the needs of the order

  def analyze_ticket_urgency
    result = TicketUrgencyAnalyzer.perform(@ticket)
    continue_with(result)
  end

  def list_available_agents
    result = ListAvailableAgents.perform
    continue_with(result)
  end

  def predict_agent_workload
    result = AgentWorkloadPredictor.perform
    continue_with(result)
  end

  def assign_ticket_to_agent
    TicketAssigner.perform(@ticket, @transcript)
  end

  def delay_assignment(until)
    until = DateTimeStandardizer.process(until)
    SupportTicketScheduler.delay(@ticket, @transcript, until)
  end

  private

```

```
  def continue_with(result)
    @transcript << { function: result }
    complete(@transcript)
  end
end
```

U ovom primeru, SupportTicketScheduler je inicijalizovan objektom tiketa za podršku i održava zapisnik raspoređivanja. AI komponenta analizira detalje tiketa i prediktivno planira dodeljivanje tiketa na osnovu faktora kao što su hitnost tiketa, stručnost agenta i predviđeno radno opterećenje agenta.

Dostupne funkcije koje AI može da pozove uključuju analyze_ticket_urgency, list_available_agents, predict_agent_workload i assign_ticket_to_agent. Svaka funkcija delegira zadatak odgovarajućoj komponenti za analizu ili predviđanje i dodaje rezultat u zapisnik raspoređivanja. AI takođe ima opciju da odloži dodeljivanje koristeći funkciju delay_assignment.

AI komponenta pregleda zapisnik raspoređivanja i donosi informisane odluke o dodeljivanju tiketa. Uzima u obzir hitnost tiketa, stručnost dostupnih agenata i predviđeno radno opterećenje svakog agenta kako bi odredila najpogodnijeg agenta za rešavanje tiketa.

Korišćenjem prediktivnog raspoređivanja zadataka, aplikacija za korisničku podršku može da optimizuje dodeljivanje tiketa, smanji vreme odziva i poboljša ukupno zadovoljstvo korisnika. Proaktivno i efikasno upravljanje tiketima za podršku osigurava da pravi tiketi budu dodeljeni pravim agentima u pravo vreme.

Upravljanje izuzecima i oporavak u procesu obrade podataka

Upravljanje izuzecima i oporavak od grešaka su ključni za osiguranje integriteta podataka i sprečavanje gubitka podataka. Komponenta DataProcessingOrchestrator koristi AI za inteligentno upravljanje izuzecima i orkestraciju procesa oporavka u procesu obrade podataka

```
class DataProcessingOrchestrator
  include Raix::ChatCompletion
  include Raix::FunctionDispatch

  SYSTEM_DIRECTIVE = "You are a data processing orchestrator..."

  def initialize(data_batch)
    @data_batch = data_batch
    @transcript = [
      { system: SYSTEM_DIRECTIVE },
      { user: data_batch.to_json }
    ]
  end

  def perform
    complete(@transcript)
  end

  def model
    "openai/gpt-4"
  end

  def functions
    [
      {
        name: "validate_data",
        # ...
      },
      {
        name: "process_data",
        # ...
      },
      {
        name: "request_fix",
        # ...
      },
      {
        name: "retry_processing",
        # ...
      },
      {
        name: "mark_data_as_failed",
```

```
43          # ...
44        },
45        {
46          name: "finished",
47          # ...
48        }
49      ]
50    end
51
52    # implementation of functions that can be called by the AI
53    # entirely at its discretion, depending on the needs of the order
54
55    def validate_data
56      result = DataValidator.perform(@data_batch)
57      continue_with(result)
58    rescue ValidationException => e
59      handle_validation_exception(e)
60    end
61
62    def process_data
63      result = DataProcessor.perform(@data_batch)
64      continue_with(result)
65    rescue ProcessingException => e
66      handle_processing_exception(e)
67    end
68
69    def request_fix(description_of_fix)
70      result = SmartDataFixer.new(description_of_fix, @data_batch)
71      continue_with(result)
72    end
73
74    def retry_processing(timeout_in_seconds)
75      wait(timeout_in_seconds)
76      process_data
77    end
78
79    def mark_data_as_failed
80      @data_batch.update!(status: 'failed', transcript: @transcript)
81    end
82
83    def finished
84      @data_batch.update!(status: 'finished', transcript: @transcript)
```

```
  end

  private

  def continue_with(result)
    @transcript << { function: result }
    complete(@transcript)
  end

  def handle_validation_exception(exception)
    @transcript << { exception: exception.message }
    complete(@transcript)
  end

  def handle_processing_exception(exception)
    @transcript << { exception: exception.message }
    complete(@transcript)
  end
end
```

U ovom primeru, DataProcessingOrchestrator je inicijalizovan objektom grupe podataka i održava transkript obrade. AI komponenta orkestira proces obrade podataka, upravlja izuzecima i oporavlja se od grešaka po potrebi.

Dostupne funkcije koje AI može da pozove uključuju validate_data, process_data, request_fix, retry_processing i mark_data_as_failed. Svaka funkcija delegira zadatak odgovarajućoj komponenti za obradu podataka i dodaje rezultat ili detalje o izuzetku u transkript obrade.

Ako se izuzetak pri validaciji dogodi tokom koraka validate_data, funkcija handle_-validation_exception dodaje podatke o izuzetku u transkript i vraća kontrolu AI-ju. Slično tome, ako se izuzetak pri obradi dogodi tokom koraka process_data, AI može da odluči o strategiji oporavka.

U zavisnosti od prirode nastalog izuzetka, AI može po sopstvenom nahođenju da odluči da pozove request_fix, koji delegira AI-pokrenutoj komponenti SmartDataFixer (pogledajte poglavlje o Samozalečujućim podacima). Program za popravku podataka

dobija opis na običnom jeziku o tome kako bi trebalo da modifikuje @data_batch tako da se obrada može ponovo pokušati. Možda bi uspešno ponovno pokušavanje podrazumevalo uklanjanje zapisa iz grupe podataka koji nisu prošli validaciju i/ili njihovo kopiranje u drugi proces obrade za ljudski pregled? Mogućnosti su skoro beskrajne.

Uključivanjem AI-vođene obrade izuzetaka i oporavka, aplikacija za obradu podataka postaje otpornija i tolerantnija na greške. DataProcessingOrchestrator inteligentno upravlja izuzecima, minimizira gubitak podataka i osigurava nesmetano izvršavanje toka obrade podataka.

Praćenje i beleženje

Praćenje i beleženje pružaju uvid u napredak, performanse i zdravlje komponenti toka rada pokrenutih AI-jem, omogućavajući programerima da prate i analiziraju ponašanje sistema. Implementacija efikasnih mehanizama za praćenje i beleženje je od suštinskog značaja za otklanjanje grešaka, reviziju i kontinuirano poboljšanje inteligentnih tokova rada.

Praćenje napretka i performansi toka rada

Da bi se osiguralo nesmetano izvršavanje inteligentnih tokova rada, važno je pratiti napredak i performanse svake komponente toka rada. Ovo uključuje praćenje ključnih metrika i događaja tokom životnog ciklusa toka rada.

Neki važni aspekti za praćenje uključuju:

1. **Vreme izvršavanja toka rada**: Merenje vremena koje je potrebno svakoj komponenti toka rada da završi svoj zadatak. Ovo pomaže u identifikovanju uskih grla u performansama i optimizaciji ukupne efikasnosti toka rada.

2. **Iskorišćenost resursa**: Praćenje iskorišćenosti sistemskih resursa, kao što su CPU, memorija i skladište, od strane svake komponente toka rada. Ovo pomaže da se osigura

da sistem radi u okviru svojih kapaciteta i da može efikasno da upravlja radnim opterećenjem.

3. **Stope grešaka i izuzeci**: Praćenje pojave grešaka i izuzetaka unutar komponenti toka rada. Ovo pomaže u identifikovanju potencijalnih problema i omogućava proaktivno rukovanje greškama i oporavak.

4. **Tačke odlučivanja i ishodi**: Praćenje tačaka odlučivanja unutar toka rada i ishoda AI-vođenih odluka. Ovo pruža uvid u ponašanje i efikasnost AI komponenti.

Podaci prikupljeni procesom praćenja mogu se prikazati na kontrolnim tablama ili koristiti kao ulazni podaci za planirane izveštaje koji informišu administratore sistema o zdravlju sistema.

Podaci praćenja mogu se proslediti AI-vođenom procesu administratora sistema na pregled i potencijalnu akciju!

Beleženje ključnih događaja i odluka

Beleženje je suštinska praksa koja uključuje hvatanje i čuvanje relevantnih informacija o ključnim događajima, odlukama i izuzecima koji se javljaju tokom izvršavanja toka rada.

Neki važni aspekti za beleženje uključuju:

1. **Pokretanje i završetak toka rada**: Beleženje vremena početka i završetka svake instance toka rada, zajedno sa svim relevantnim metapodacima kao što su ulazni podaci i korisnički kontekst.

2. **Izvršavanje komponenti**: Beleženje detalja izvršavanja svake komponente toka rada, uključujući ulazne parametre, izlazne rezultate i sve generisane međupodatke.

3. **AI odluke i obrazloženja**: Beleženje odluka koje donose AI komponente, zajedno sa osnovnim obrazloženjem ili ocenama pouzdanosti. Ovo pruža transparentnost i omogućava reviziju AI-vođenih odluka.

4. **Izuzeci i poruke o greškama**: Beleženje svih izuzetaka ili poruka o greškama na koje se naiđe tokom izvršavanja toka rada, uključujući praćenje steka i relevantne kontekstualne informacije.

Beleženje se može implementirati koristeći različite tehnike, kao što su pisanje u datoteke evidencije, čuvanje evidencija u bazi podataka ili slanje evidencija centralizovanoj usluzi za beleženje. Važno je odabrati okvir za beleženje koji pruža fleksibilnost, skalabilnost i laku integraciju sa arhitekturom aplikacije.

Evo primera kako se beleženje može implementirati u Ruby on Rails aplikaciji koristeći klasu ActiveSupport::Logger:

```
class WorkflowLogger
  def self.log(message, severity = :info)
    @logger ||= ActiveSupport::Logger.new('workflow.log')
    @logger.formatter ||= proc do |severity, datetime, progname, msg|
      "#{datetime} [#{severity}] #{msg}\n"
    end
    @logger.send(severity, message)
  end
end

# Usage example
WorkflowLogger.log("Workflow initiated for order ##{@order.id}")
WorkflowLogger.log("Payment processing completed successfully")
WorkflowLogger.log("Inventory check failed for item ##{item.id}", :error)
```

Strateškim postavljanjem izraza za beleženje kroz komponente radnog toka i tačke odlučivanja veštačke inteligencije, programeri mogu da zabeleže vredne informacije za otklanjanje grešaka, reviziju i analizu.

Prednosti praćenja i beleženja

Implementacija praćenja i beleženja u inteligentnoj orkestraciji radnih tokova nudi nekoliko prednosti:

1. **Otklanjanje grešaka i rešavanje problema**: Detaljni zapisi i podaci praćenja pomažu programerima da brzo identifikuju i dijagnostikuju probleme. Oni pružaju uvid u tok izvršavanja radnog toka, interakcije komponenti i sve greške ili izuzetke na koje se naiđe.

2. **Optimizacija performansi**: Praćenje metrika performansi omogućava programerima da identifikuju uska grla i optimizuju komponente radnog toka za bolju efikasnost. Analiziranjem vremena izvršavanja, iskorišćenosti resursa i drugih metrika, programeri mogu donositi informisane odluke za poboljšanje ukupnih performansi sistema.

3. **Revizija i usklađenost**: Beleženje ključnih događaja i odluka obezbeđuje revizorski trag za regulatornu usklađenost i odgovornost. Omogućava organizacijama da prate i verifikuju akcije koje preduzimaju AI komponente i osiguraju pridržavanje poslovnih pravila i zakonskih zahteva.

4. **Kontinuirano poboljšanje**: Podaci praćenja i beleženja služe kao vredni ulazni podaci za kontinuirano poboljšanje inteligentnih radnih tokova. Analiziranjem istorijskih podataka, identifikovanjem obrazaca i merenjem efektivnosti AI odluka, programeri mogu iterativno da usavršavaju i unapređuju logiku orkestracije radnog toka.

Razmatranja i najbolje prakse

Pri implementaciji praćenja i beleženja u inteligentnoj orkestraciji radnih tokova, razmotrite sledeće najbolje prakse:

1. **Definisanje jasnih metrika praćenja**: Identifikujte ključne metrike i događaje koje treba pratiti na osnovu specifičnih zahteva radnog toka. Fokusirajte se na metrike koje pružaju smislene uvide u performanse, zdravlje i ponašanje sistema.

2. **Implementacija granularnog beleženja**: Osigurajte da su izrazi za beleženje postavljeni na odgovarajućim tačkama unutar komponenti radnog toka i tačaka odlučivanja AI-ja. Zabeležite relevantne kontekstualne informacije, kao što su ulazni parametri, izlazni rezultati i svi međupodaci koji se generišu.

3. **Korišćenje strukturiranog beleženja**: Usvojite format strukturiranog beleženja kako biste olakšali jednostavno parsiranje i analizu podataka iz zapisa. Strukturirano beleženje omogućava bolju pretraživost, filtriranje i agregaciju zapisa.

4. **Upravljanje zadržavanjem i rotacijom zapisa**: Implementirajte politike zadržavanja i rotacije zapisa za upravljanje skladištenjem i životnim ciklusom datoteka zapisa. Odredite odgovarajući period zadržavanja na osnovu zakonskih zahteva, ograničenja skladištenja i potreba analize. Ako je moguće, prebacite beleženje na uslugu treće strane kao što je Papertrail.

5. **Zaštita osetljivih informacija**: Budite oprezni pri beleženju osetljivih informacija, kao što su lični podaci (PII) ili poverljivi poslovni podaci. Implementirajte odgovarajuće sigurnosne mere, kao što su maskiranje podataka ili enkripcija, kako biste zaštitili osetljive informacije u datotekama zapisa.

6. **Integracija sa alatima za praćenje i upozoravanje**: Iskoristite alate za praćenje i upozoravanje za centralizaciju prikupljanja, analize i vizualizacije podataka praćenja i beleženja. Ovi alati mogu pružiti uvide u realnom vremenu, generisati upozorenja na osnovu unapred definisanih pragova i olakšati proaktivno otkrivanje i rešavanje problema. Moj omiljeni od ovih alata je Datadog.

Implementacijom sveobuhvatnih mehanizama praćenja i beleženja, programeri mogu dobiti vredne uvide u ponašanje i performanse inteligentnih radnih tokova. Ovi uvidi omogućavaju efikasno otklanjanje grešaka, optimizaciju i kontinuirano poboljšanje sistema za orkestraciju radnih tokova zasnovanih na veštačkoj inteligenciji.

Razmatranja skalabilnosti i performansi

Skalabilnost i performanse su kritični aspekti koje treba razmotriti pri dizajniranju i implementaciji sistema za inteligentnu orkestraciju radnih tokova. Kako se povećava obim istovremenih radnih tokova i složenost komponenti zasnovanih na veštačkoj

inteligenciji, postaje neophodno osigurati da sistem može efikasno da upravlja radnim opterećenjem i nesmetano se skalira kako bi zadovoljio rastuće zahteve.

Upravljanje velikim obimom istovremenih radnih tokova

Sistemi za inteligentnu orkestraciju radnih tokova često moraju da upravljaju velikim brojem istovremenih radnih tokova. Da biste osigurali skalabilnost, razmotrite sledeće strategije:

1. **Asinhrona obrada**: Implementirajte mehanizme asinhrone obrade za razdvajanje izvršavanja komponenti radnog toka. Ovo omogućava sistemu da upravlja sa više radnih tokova istovremeno bez blokiranja ili čekanja da se svaka komponenta završi. Asinhrona obrada se može postići korišćenjem redova poruka, arhitektura vođenih događajima ili radnih okvira za obradu pozadinskih poslova kao što je Sidekiq.

2. **Distribuirana arhitektura**: Dizajnirajte arhitekturu sistema tako da koristi serverless komponente (kao što je AWS Lambda) ili jednostavno distribuirajte radno opterećenje preko više čvorova ili servera zajedno sa vašim glavnim aplikacionim serverom. Ovo omogućava horizontalnu skalabilnost, gde se dodatni čvorovi mogu dodati za upravljanje povećanim obimom radnih tokova.

3. **Paralelno izvršavanje**: Identifikujte mogućnosti za paralelno izvršavanje unutar radnih tokova. Neke komponente radnog toka mogu biti nezavisne jedna od druge i mogu se izvršavati istovremeno. Korišćenjem tehnika paralelne obrade, kao što su višenitnost ili distribuirani redovi zadataka, sistem može optimizovati korišćenje resursa i smanjiti ukupno vreme izvršavanja radnog toka.

Optimizacija performansi AI komponenti

AI komponente, kao što su modeli mašinskog učenja ili sistemi za obradu prirodnog jezika, mogu biti računarski zahtevne i uticati na ukupne performanse sistema

za orkestraciju radnih tokova. Da biste optimizovali performanse AI komponenti, razmotrite sledeće tehnike:

1. **Keširanje**: Ako je vaša AI obrada čisto generativna i ne uključuje pretraživanje informacija u realnom vremenu ili eksterne integracije za generisanje chat odgovora, možete istražiti mehanizme keširanja za čuvanje i ponovno korišćenje rezultata često pristupanih ili računarski zahtevnih operacija.

2. **Optimizacija modela**: Kontinuirano optimizujte način na koji koristite AI modele u komponentama radnog toka. Ovo može uključivati tehnike kao što je *Destilacija promptova* ili jednostavno testiranje novih modela kako postaju dostupni.

3. **Grupna obrada**: Ako radite sa modelima klase GPT-4, možda ćete moći da iskoristite tehnike grupne obrade za obradu više podataka ili zahteva u jednoj grupi, umesto njihove pojedinačne obrade. Obradom podataka u grupama, sistem može optimizovati korišćenje resursa i smanjiti opterećenje višestrukih zahteva modelu.

Praćenje i profilisanje performansi

Za identifikaciju uskih grla u performansama i optimizaciju skalabilnosti inteligentnog sistema za orkestraciju radnih tokova, ključno je implementirati mehanizme za praćenje i profilisanje. Razmotrite sledeće pristupe:

1. **Metrike performansi**: Definišite i pratite ključne metrike performansi, kao što su vreme odziva, propusnost, iskorišćenost resursa i kašnjenje. Ove metrike pružaju uvid u performanse sistema i pomažu u identifikaciji područja za optimizaciju. Popularni AI model agregator OpenRouter uključuje Host[1] i Speed[2] metrike u svakom API odgovoru, čineći praćenje ovih ključnih metrika trivijalnim.

2. **Alati za profilisanje**: Koristite alate za profilisanje kako biste analizirali performanse pojedinačnih komponenti radnog toka i AI operacija. Alati za profilisanje mogu pomoći

[1]Host je vreme potrebno za primanje prvog bajta streamovanog generisanja od strane host-a modela, poznatog i kao "vreme do prvog bajta."

[2]Speed se izračunava kao broj tokena za dovršavanje podeljen sa ukupnim vremenom generisanja. Za zahteve koji nisu streamovani, kašnjenje se smatra delom vremena generisanja.

u identifikaciji kritičnih tačaka performansi, neefikasnih putanja koda ili operacija koje intenzivno koriste resurse. Popularni alati za profilisanje uključuju New Relic, Scout, ili ugrađene profilere koje pruža programski jezik ili framework.

3. **Testiranje opterećenja**: Sprovedite testiranje opterećenja kako biste procenili performanse sistema pod različitim nivoima istovremenih radnih opterećenja. Testiranje opterećenja pomaže u identifikaciji granica skalabilnosti sistema, otkrivanju degradacije performansi i osiguravanju da sistem može podneti očekivani saobraćaj bez ugrožavanja performansi.

4. **Kontinuirano praćenje**: Implementirajte mehanizme kontinuiranog praćenja i upozoravanja kako biste proaktivno otkrili probleme sa performansama i uska grla. Postavite kontrolne table za praćenje i upozorenja za praćenje ključnih indikatora performansi (KPI) i primanje obaveštenja kada se prekorače unapred definisani pragovi. Ovo omogućava brzu identifikaciju i rešavanje problema sa performansama.

Strategije skaliranja

Da biste upravljali povećanim radnim opterećenjem i osigurali skalabilnost inteligentnog sistema za orkestraciju radnih tokova, razmotrite sledeće strategije skaliranja:

1. **Vertikalno skaliranje**: Vertikalno skaliranje uključuje povećanje resursa (npr. CPU, memorija) pojedinačnih čvorova ili servera za rukovanje većim radnim opterećenjem. Ovaj pristup je pogodan kada sistem zahteva više procesorske snage ili memorije za rukovanje složenim radnim tokovima ili AI operacijama.

2. **Horizontalno skaliranje**: Horizontalno skaliranje uključuje dodavanje više čvorova ili servera sistemu za distribuciju radnog opterećenja. Ovaj pristup je efikasan kada sistem treba da upravlja velikim brojem istovremenih radnih tokova ili kada se radno opterećenje može lako distribuirati preko više čvorova. Horizontalno skaliranje zahteva distribuiranu arhitekturu i mehanizme balansiranja opterećenja kako bi se osigurala ravnomerna distribucija saobraćaja.

3. **Automatsko skaliranje**: Implementirajte mehanizme automatskog skaliranja za automatsko prilagođavanje broja čvorova ili resursa na osnovu zahteva radnog opterećenja. Automatsko skaliranje omogućava sistemu da se dinamički skalira gore ili dole u zavisnosti od dolaznog saobraćaja, osiguravajući optimalnu iskorišćenost resursa i ekonomičnost. Cloud platforme poput Amazon Web Services (AWS) ili Google Cloud Platform (GCP) pružaju mogućnosti automatskog skaliranja koje se mogu iskoristiti za inteligentne sisteme orkestracije radnih tokova.

Tehnike optimizacije performansi

Pored strategija skaliranja, razmotrite sledeće tehnike optimizacije performansi za poboljšanje efikasnosti inteligentnog sistema za orkestraciju radnih tokova:

1. **Efikasno skladištenje i preuzimanje podataka**: Optimizujte mehanizme skladištenja i preuzimanja podataka koje koriste komponente radnog toka. Koristite efikasno indeksiranje baze podataka, tehnike optimizacije upita i keširanje podataka kako biste smanjili kašnjenje i poboljšali performanse operacija intenzivnih podataka.

2. **Asinhroni U/I**: Koristite asinhrone U/I operacije kako biste sprečili blokiranje i poboljšali odziv sistema. Asinhroni U/I omogućava sistemu da istovremeno obrađuje više zahteva bez čekanja na završetak U/I operacija, čime se maksimalno iskorišćavaju resursi.

3. **Efikasna serijalizacija i deserijalizacija**: Optimizujte procese serijalizacije i deserijalizacije koji se koriste za razmenu podataka između komponenti radnog toka. Koristite efikasne formate serijalizacije, kao što su Protocol Buffers ili MessagePack, kako biste smanjili opterećenje serijalizacije podataka i poboljšali performanse komunikacije između komponenti.

Za aplikacije zasnovane na Ruby-ju, razmotrite korišćenje Universal ID. Universal ID koristi i MessagePack i Brotli (kombinaciju napravljenu za brzinu i najbolju kompresiju podataka u klasi). Kada se kombinuju, ove biblioteke su do 30% brže i imaju stope kompresije koje su za samo 2-5% lošije u poređenju sa Protocol Buffers.

4. **Kompresija i kodiranje**: Primenite tehnike kompresije i kodiranja kako biste smanjili veličinu podataka koji se prenose između komponenti radnog toka. Algoritmi za kompresiju, kao što su gzip ili Brotli, mogu značajno smanjiti korišćenje mrežnog propusnog opsega i poboljšati ukupne performanse sistema.

Uzimajući u obzir aspekte skalabilnosti i performansi tokom dizajna i implementacije sistema za inteligentnu orkestraciju radnih tokova, možete osigurati da vaš sistem može da upravlja velikim obimom istovremenih radnih tokova, optimizuje performanse komponenti zasnovanih na veštačkoj inteligenciji i nesmetano se skalira kako bi zadovoljio rastuće zahteve. Kontinuirano praćenje, profilisanje i napori za optimizaciju su ključni za održavanje performansi i odzivnosti sistema kako se opterećenje i složenost povećavaju tokom vremena.

Testiranje i validacija radnih tokova

Testiranje i validacija su ključni aspekti razvoja i održavanja sistema za inteligentnu orkestraciju radnih tokova. S obzirom na složenu prirodu radnih tokova zasnovanih na veštačkoj inteligenciji, neophodno je osigurati da svaka komponenta funkcioniše kako se očekuje, da se celokupni radni tok ponaša ispravno i da su odluke veštačke inteligencije tačne i pouzdane. U ovom odeljku ćemo istražiti različite tehnike i razmatranja za testiranje i validaciju inteligentnih radnih tokova.

Jedinično testiranje komponenti radnog toka

Jedinično testiranje podrazumeva testiranje pojedinačnih komponenti radnog toka izolovano kako bi se proverila njihova ispravnost i robusnost. Prilikom jediničnog testiranja komponenti zasnovanih na veštačkoj inteligenciji, razmotrite sledeće:

1. **Validacija ulaznih podataka**: Testirajte sposobnost komponente da obrađuje različite tipove ulaznih podataka, uključujući validne i nevalidne podatke. Proverite da li komponenta elegantno upravlja graničnim slučajevima i pruža odgovarajuće poruke o greškama ili izuzetke.

2. **Verifikacija izlaznih podataka**: Potvrdite da komponenta proizvodi očekivani izlaz za dati skup ulaznih podataka. Uporedite stvarni izlaz sa očekivanim rezultatima kako biste osigurali ispravnost.

3. **Upravljanje greškama**: Testirajte mehanizme za upravljanje greškama komponente simuliranjem različitih scenarija grešaka, kao što su nevažeći ulazni podaci, nedostupnost resursa ili neočekivani izuzeci. Proverite da li komponenta ispravno hvata i upravlja greškama.

4. **Granični uslovi**: Testirajte ponašanje komponente pod graničnim uslovima, kao što su prazan ulaz, maksimalna veličina ulaza ili ekstremne vrednosti. Osigurajte da komponenta elegantno upravlja ovim uslovima bez rušenja ili proizvodnje netačnih rezultata.

Evo primera jediničnog testa za komponentu radnog toka u Ruby-ju koristeći RSpec radni okvir za testiranje:

```
RSpec.describe OrderValidator do
  describe '#validate' do
    context 'when order is valid' do
      let(:order) { build(:order) }

      it 'returns true' do
        expect(subject.validate(order)).to be true
      end
    end

    context 'when order is invalid' do
      let(:order) { build(:order, total_amount: -100) }

      it 'returns false' do
        expect(subject.validate(order)).to be false
      end
    end
  end
end
```

U ovom primeru, OrderValidator komponenta se testira korišćenjem dva test slučaja: jedan za validan nalog i drugi za nevalidan nalog. Test slučajevi verifikuju da metoda validate vraća očekivanu bulovu vrednost na osnovu validnosti naloga.

Integraciono testiranje interakcija toka rada

Integraciono testiranje se fokusira na verifikaciju interakcija i toka podataka između različitih komponenti toka rada. Ono osigurava da komponente rade zajedno bez problema i proizvode očekivane rezultate. Prilikom integracionog testiranja inteligentnih tokova rada, uzmite u obzir sledeće:

1. **Interakcija komponenti**: Testirajte komunikaciju i razmenu podataka između komponenti toka rada. Verifikujte da se izlaz jedne komponente ispravno prosleđuje kao ulaz sledećoj komponenti u toku rada.

2. **Konzistentnost podataka**: Osigurajte da podaci ostaju konzistentni i tačni dok prolaze kroz tok rada. Verifikujte da se transformacije podataka, kalkulacije i agregacije

izvršavaju ispravno.

3. **Propagacija izuzetaka**: Testirajte kako se izuzeci i greške propagiraju i obrađuju kroz komponente toka rada. Verifikujte da su izuzeci uhvaćeni, zabeleženi i pravilno obrađeni kako bi se sprečio prekid toka rada.

4. **Asinhrono ponašanje**: Ako tok rada uključuje asinhrone komponente ili paralelno izvršavanje, testirajte mehanizme koordinacije i sinhronizacije. Osigurajte da se tok rada ponaša ispravno u konkurentnim i asinhronim scenarijima.

Evo primera integracionog testa za tok rada u Ruby-ju koristeći RSpec radni okvir za testiranje:

```
RSpec.describe OrderProcessingWorkflow do

  let(:order) { build(:order) }

  it 'processes the order successfully' do
    expect(OrderValidator).to receive(:validate).and_return(true)
    expect(InventoryManager).to receive(:check_availability).and_return(true)
    expect(PaymentProcessor).to receive(:process_payment).and_return(true)
    expect(ShippingService).to receive(:schedule_shipping).and_return(true)

    workflow = OrderProcessingWorkflow.new(order)
    result = workflow.process

    expect(result).to be true
    expect(order.status).to eq('processed')
  end

end
```

U ovom primeru, OrderProcessingWorkflow se testira proverom interakcija između različitih komponenti toka rada. Test postavlja očekivanja za ponašanje svake komponente i osigurava da tok rada uspešno obrađuje narudžbinu, ažurirajući status narudžbine u skladu sa tim.

Testiranje AI tačaka odlučivanja

Testiranje AI tačaka odlučivanja je ključno za osiguravanje preciznosti i pouzdanosti tokova rada zasnovanih na veštačkoj inteligenciji. Prilikom testiranja AI tačaka odlučivanja, razmotrite sledeće:

1. **Preciznost odlučivanja**: Proverite da AI komponenta donosi precizne odluke na osnovu ulaznih podataka i treniranog modela. Uporedite AI odluke sa očekivanim ishodima ili referentnim podacima.

2. **Granični slučajevi**: Testirajte ponašanje AI komponente u graničnim slučajevima i neuobičajenim scenarijima. Proverite da li AI komponenta elegantno upravlja ovim slučajevima i donosi razumne odluke.

3. **Pristrasnost i pravičnost**: Procenite AI komponentu na potencijalne pristrasnosti i osigurajte da donosi pravične i nepristrasne odluke. Testirajte komponentu sa raznovrsnim ulaznim podacima i analizirajte ishode u potrazi za diskriminatornim obrascima.

4. **Objašnjivost**: Ako AI komponenta pruža objašnjenja ili obrazloženja za svoje odluke, proverite tačnost i jasnoću objašnjenja. Osigurajte da su objašnjenja usklađena sa osnovnim procesom donošenja odluka.

Evo primera testiranja AI tačke odlučivanja u Ruby-ju koristeći RSpec okvir za testiranje:

```
RSpec.describe FraudDetector do
  describe '#detect_fraud' do
    context 'when transaction is fraudulent' do
      let(:tx) do
        build(:transaction, amount: 10_000, location: 'High-Risk Country')
      end

      it 'returns true' do
        expect(subject.detect_fraud(tx)).to be true
      end
    end

    context 'when transaction is legitimate' do
      let(:tx) do
        build(:transaction, amount: 100, location: 'Low-Risk Country')
      end

      it 'returns false' do
        expect(subject.detect_fraud(tx)).to be false
      end
    end
  end
end
```

U ovom primeru, FraudDetector AI komponenta je testirana sa dva test primera: jedan za lažnu transakciju i drugi za legitimnu transakciju. Test primeri proveravaju da li detect_fraud metoda vraća očekivanu bulovsku vrednost na osnovu karakteristika transakcije.

Testiranje s kraja na kraj

Testiranje s kraja na kraj podrazumeva testiranje celokupnog toka rada od početka do kraja, simulirajući scenarije iz stvarnog sveta i korisničke interakcije. Ono osigurava da se tok rada ponaša ispravno i proizvodi željene rezultate. Prilikom izvođenja testiranja s kraja na kraj za inteligentne tokove rada, uzmite u obzir sledeće:

1. **Korisnički scenariji**: Identifikujte uobičajene korisničke scenarije i testirajte

ponašanje toka rada u tim scenarijima. Proverite da li tok rada pravilno obrađuje korisničke unose, donosi odgovarajuće odluke i proizvodi očekivane rezultate.

2. **Validacija podataka**: Osigurajte da tok rada validira i prečišćava korisničke unose kako bi se sprečile nekonzistentnosti podataka ili bezbednosne ranjivosti. Testirajte tok rada sa različitim tipovima ulaznih podataka, uključujući validne i nevalidne podatke.

3. **Oporavak od grešaka**: Testirajte sposobnost toka rada da se oporavi od grešaka i izuzetaka. Simulirajte scenarije grešaka i proverite da li tok rada elegantno upravlja njima, beleži greške i preduzima odgovarajuće akcije oporavka.

4. **Performanse i skalabilnost**: Procenite performanse i skalabilnost toka rada pod različitim uslovima opterećenja. Testirajte tok rada sa velikim obimom konkurentnih zahteva i merite vreme odziva, iskorišćenost resursa i ukupnu stabilnost sistema.

Evo primera testa s kraja na kraj za tok rada u Ruby-ju koristeći RSpec okvir za testiranje i Capybara biblioteku za simulaciju korisničkih interakcija:

```
RSpec.describe 'Order Processing Workflow' do
  scenario 'User places an order successfully' do
    visit '/orders/new'
    fill_in 'Product', with: 'Sample Product'
    fill_in 'Quantity', with: '2'
    fill_in 'Shipping Address', with: '123 Main St'
    click_button 'Place Order'

    expect(page).to have_content('Order Placed Successfully')
    expect(Order.count).to eq(1)
    expect(Order.last.status).to eq('processed')
  end
end
```

U ovom primeru, test s kraja na kraj simulira korisnika koji postavlja narudžbinu preko web interfejsa. Popunjava potrebna polja obrasca, šalje narudžbinu i proverava da li je narudžbina uspešno obrađena, prikazujući odgovarajuću poruku potvrde i ažurirajući status narudžbine u bazi podataka.

Kontinuirana integracija i isporuka

Da bi se osigurala pouzdanost i održivost inteligentnih tokova rada, preporučuje se integracija testiranja i validacije u protok kontinuirane integracije i isporuke (CI/CD). Ovo omogućava automatizovano testiranje i validaciju promena u toku rada pre nego što se primene u produkciji. Razmotrite sledeće prakse:

1. **Automatizovano izvršavanje testova**: Konfigurišite CI/CD protok da automatski pokreće paket testova kad god se naprave izmene u kodu toka rada. Ovo osigurava da se sve regresije ili greške otkriju rano u procesu razvoja.

2. **Praćenje pokrivenosti testovima**: Merite i pratite pokrivenost testovima komponenti toka rada i AI tačaka odlučivanja. Težite visokoj pokrivenosti testovima kako biste osigurali da su kritične putanje i scenariji temeljno testirani.

3. **Kontinuirana povratna informacija**: Integrišite rezultate testova i metrike kvaliteta koda u tok razvoja. Obezbedite kontinuiranu povratnu informaciju programerima o statusu testova, kvalitetu koda i svim problemima otkrivenim tokom CI/CD procesa.

4. **Razvojna okruženja**: Primenite tok rada u razvojnim okruženjima koja blisko odslikavaju produkciono okruženje. Izvršite dodatno testiranje i validaciju u razvojnom okruženju kako biste otkrili sve probleme vezane za infrastrukturu, konfiguraciju ili integraciju podataka.

5. **Mehanizmi za povratak**: Implementirajte mehanizme za povratak u slučaju neuspeha pri primeni ili kritičnih problema otkrivenih u produkciji. Osigurajte da se tok rada može brzo vratiti na prethodnu stabilnu verziju kako bi se minimiziralo vreme prekida rada i uticaj na korisnike.

Uključivanjem testiranja i validacije tokom celog životnog ciklusa razvoja inteligentnih tokova rada, organizacije mogu osigurati pouzdanost, tačnost i održivost svojih AI

sistema. Redovno testiranje i validacija pomažu u otkrivanju grešaka, sprečavanju regresija i izgradnji poverenja u ponašanje i rezultate toka rada.

Deo 2: Obrasci

Inženjerstvo promptova

Lanac razmišljanja

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

Kako funkcioniše

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

Primeri

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

Generisanje sadržaja

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

Kreiranje strukturiranih entiteta

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

Usmeravanje LLM Agenta

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

Prednosti i razmatranja

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

Promena režima

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

Kako funkcioniše

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

Kada ga koristiti

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

Primer

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

Dodela uloge

Kako funkcioniše

Kada ga koristiti

Primeri

Prompt Object

Kako funkcioniše

Šablon upita

Kako funkcioniše

Prednosti i razmatranja

Kada ga koristiti:

Primer

Strukturirani UI/IZ

Kako Funkcioniše

Skaliranje strukturiranog UI/IZ

Prednosti i razmatranja

Ulančavanje promptova

Kako funkcioniše

Kada ga koristiti

Primer: Olimpijino uvođenje korisnika

Prepisivač Promptova

Kako Funkcioniše

Primer

Ograđivanje odgovora

Kako funkcioniše

Prednosti i razmatranja

Rukovanje greškama

Analizator upita

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

Kako funkcioniše

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

Implementacija

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

Označavanje vrsta reči (POS) i Prepoznavanje imenovanih entiteta (NER)

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

Klasifikacija namere

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

Ekstrakcija ključnih reči

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

Prednosti

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

Prepisivač upita

Kako funkcioniše

Primer

Prednosti

Ventrilokvist

Kako Funkcioniše

Kada Ga Koristiti

Primer

Diskretne komponente

Predikat

Kako funkcioniše

Kada ga koristiti

Primer

API Fasada

Kako funkcioniše

Ključne prednosti

Kada je koristiti

Primer

Autentifikacija i Autorizacija

Obrada Zahteva

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

Formatiranje Odgovora

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

Obrada Grešaka i Graničnih Slučajeva

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

Razmatranja o Skalabilnosti i Performansama

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

Poređenje sa Drugim Dizajn Obrascima

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

Interpreter rezultata

Kako funkcioniše

Kada ga koristiti

Primer

Virtuelna mašina

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

Kako funkcioniše

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

Kada ga koristiti

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

Primer

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

Iza Magije

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

Specifikacija i Testiranje

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

Specifikacija Ponašanja

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

Pisanje Test Primera

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

Primer: Testiranje Translator Komponente

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

Reprodukcija HTTP Interakcija

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

Čovek u petlji (HITL)

Obrasci visokog nivoa

Hibridna inteligencija

Adaptivni odziv

Zamena uloga između čoveka i AI

Eskalacija

Kako funkcioniše

Ključne prednosti

Primena u stvarnom svetu: Zdravstvo

Povratna sprega

Kako to funkcioniše

Primene i Primeri

Napredne Tehnike u Integraciji Ljudskih Povratnih Informacija

Pasivno zračenje informacija

Kako funkcioniše

Kontekstualni prikaz informacija

Proaktivna obaveštenja

Objašnjavajući uvidi

Interaktivno istraživanje

Ključne prednosti

Primene i primeri

Kolaborativno donošenje odluka (CDM)

Kako funkcioniše

Primer

Kontinuirano učenje

Kako funkcioniše

Primene i primeri

Primer

Etička razmatranja

Uloga HITL-a u ublažavanju AI rizika

Tehnološki napredak i budući izgledi

Izazovi i ograničenja HITL sistema

Inteligentno rukovanje greškama

Tradicionalni pristupi rukovanju greškama

Kontekstualna dijagnostika grešaka

Kako funkcioniše

Inženjerstvo promptova za kontekstualnu dijagnostiku grešaka

Generisanje Potpomognuto Preuzimanjem za Kontekstualnu Dijagnostiku Grešaka

Inteligentno Izveštavanje o Greškama

Prediktivno sprečavanje grešaka

Kako Funkcioniše

Pametni Oporavak od Grešaka

Kako Funkcioniše

Personalizovana komunikacija o greškama

Kako funkcioniše

Adaptivni tok rada za rukovanje greškama

Kako funkcioniše

Kontrola kvaliteta

Eval

Problem

Rešenje

Kako funkcioniše

Primer

Razmatranja

Razumevanje zlatnih referenci

Kako funkcionišu evaluacije bez reference

Guardrail

Problem

Rešenje

Kako funkcioniše

Primer

Razmatranja

Zaštitne mere i Evaluacije: Dve strane istog novčića

Međusobna zamenjivost Zaštitnih mera i Evaluacija bez referenci

Implementacija Zaštitnih mera i Evaluacija sa dvostrukom namenom

Rečnik pojmova

Rečnik pojmova

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

A

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

B

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

C

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

D

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

E

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

F

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

G

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

H

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

I

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

J

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

K

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

L

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

M

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

N

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

O

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

P

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

Q

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

R

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

S

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

T

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

U

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

V

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

W

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

Z

Ovaj sadržaj nije dostupan u uzorku knjige. Knjigu možete kupiti na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-sr-Latn.

Index

ACID svojstva, 102
adaptivni tok rada
    Adaptivna Kompozicija Toka Rada, 210
adaptivni UI, 193
Agentni, 30
AI, 60, 92, 133, 139, 188, 195
    aplikacije, 117, 129
    konverzacijska, 196
    konverzacijski, 29
    konverzaciona, 6
    model, 83, 92, 144, 145, 147, 195
    složeni sistemi, 28, 32
    tačke odlučivanja, 240
Alpaca, 12
Altman, Sam, 16
Amazon Web Services, 235
analiza sentimenta, 15, 93, 104--106, 109, 110, 126, 135
Analiza Sentimenta Kupaca, 92
ansambli, 109, 110
    ansambl radnika, 110
Anthropic, 21, 36, 68, 120, 128
antropomorfizam, 64
API-ji, 66, 142
APIs, 115
Arhitektura mikroservisa, 83
arhitektura poslovnih aplikacija, 35
arhitektura vođena događajima, 101
asinhrona obrada, 232
auto-regressive modeling, 40
Automatski Nastavak, 148
automatsko skaliranje, 235

baze podataka, 115
    -podržani objekat, 98
    strategije zaključavanja, 102
baze znanja, 7
BERT, 12, 22
bez stanja, 145
Brotli, 236
Byte Pair Encoding (BPE), 13

C (Programski jezik), 108
Capybara biblioteka, 242
Chain of Thought (CoT), 129
ChatGPT, 28, 49
Claude, 7, 40, 72
Claude 3, 46, 118, 120, 126, 128
Claude 3 Opus, 69
Claude v1, 15
Claude v2, 15
Cohere (LLM Provider), 21, 23

command line
    Command-Line Interface (CLI), 23
content
    filtering, 24
context
    infinitely long inputs, 14
    window, 14
conversation
    transkript, 145
customization, 24

data
    privacy, 24
Databricks zaposleni, 48
Datadog, 231
debagovanje
    i testiranje, 123
desktop računari, 203
detekcija prevare
    sistem, 90
determinističko ponašanje, 54
digitalno okruženje, 179
Dinamički odabir alata, 122
dinamičko generisanje korisničkog
        interfejsa, 174
Dinamičko usmeravanje zadataka, 208
distribuirana arhitektura, 232
dizajn aplikacija i radni okviri, 184
dodeljivanje tiketa, 223
Dohan, et al., 40

e-trgovina, 177
efikasnost, 207
ekosistem, 137
eksperimentisanje
    radni okvir, 180
eksterni servisi ili API-ji, 118
elektronska trgovina, 206
ELK stack, 103
emocionalni ton, 135
errors
    Intelligent Error Handling, 133
etika
    implikacije, 185

F#, 86
Facebook, 22
faktori rizika, 89, 90
filtriranje zasnovano na sadržaju, 85
finalize metoda, 145--147
fino podešavanje, 74
FitAI, 196
fleksibilnost i kreativnost, 182
funkcija
    imena, 143
    istorija poziva, 145
    pozivanje, 115
funkcionalno programiranje, 85

Gemma 7B, 10
Generative Pre-trained Transformer (GPT),
        7
Generativni korisnički interfejs (GenUI),
        198
Generativni prethodno obučeni
        transformator (GPT), 62

Generativni UI (GenUI), 184, 191, 194, 202
Generisanje potpomognuto pretraživanjem (RAG), 117
Generisanje potpomognuto preuzimanjem (RAG), 29, 35, 42, 74
generisanje sintetičkih podataka, 49
GitLab, 86
Global Interpreter Lock (GIL), 107
Google, 21
  API, 58, 60
  Cloud AI Platform, 22
  Cloud Platform, 235
  Gemini, 19
  Gemini 1.5 Pro, 12, 16, 17
  PaLM (Pathways Language Model), 16, 22
  T5, 12
GPT-3, 12, 15
GPT-4, 6, 12, 15, 16, 19, 29, 40, 46, 58, 97, 109, 112, 119, 124, 189, 190, 233
grafički modeli, 40
Graham, Paul, 17
gramatička pravila, 4
granični slučajevi, 54
granični uslovi, 237
granularno beleženje, 230
GraphQL, 100
greške
  oporavak, 242
  rukovanje, 99, 132
  stope, 103
  upravljanje, 102, 237
Groq, 24, 112
grupisanje dokumenata, 112
grupna obrada, 233
gzip, 236

hardware, 26
heš, 141
high-performance completion, 24
hiperparametar, 43
Hohpe, Gregor, 97
Honeybadger, 87
HTTP, 139

identifikacija tema, 112
informacije
  ekstrakcija, 48
  preuzimanje, 117
  pronalaženje, 6
inkluzivni interfejsi, 185
integraciono testiranje, 238
integrisanje LLM-ova, 174
inteligentna orkestracija radnih tokova, 205, 233, 236
Inteligentni moderator sadržaja, 217
interakcije u stilu igranja uloga, 6
interfejsi kojima se upravlja glasom, 31
internacionalizacija, 180
Interpreter rezultata, 132
istorijski obrasci, 209
istovremeni radni tokovi, 236
iterativno refinement, 134
iterativno usavršavanje, 70
izgradnja narativa, 18

jezik
-povezani zadaci, 4
Detekcija jezika, 104
modeli, 67
JSON (JavaScript Object Notation), 118, 122, 126, 137, 154

K-sredine, 113
kazne za ponavljanje, 47
Kazneni faktor prisustva, 45
keširanje, 233
klasifikacija, 48, 112
ključni obrasci, 208
Kodiranje parova bajtova (BPE), 12
kolaborativno filtriranje, 85
konceptualni i praktični izazovi, 185
kontekst
Kontekstualni predlozi polja, 186
kontekstualno donošenje odluka, 209
Kontekstualno generisanje sadržaja, 173, 177--179, 185, 186
prozor, 209
Proširenje, 42
Kontinuirana integracija i isporuka (CI/CD), 243
protok, 243
Kontinuirano praćenje rizika, 96
konzistentnost
i reproduktivnost, 124
korisnička podrška, 30
korisnički generisan sadržaj, 104
Korisnički interfejs (UI)
dizajn, 203
interfejsi, 184, 198
okviri, 199
tehnologije, 194
korisničko iskustvo, 180
korisničko testiranje i povratne informacije, 182
kreativno pisanje, 32, 48
Kvantizacija, 26

Lanac razmišljanja (CoT), 41
lanac snabdevanja
optimizacija, 30
language
models, 39, 61
Large Language Model (LLM), 14, 134
landscape, 25
latencija, 25
Latentna Dirihleova alokacija, 113
latentni prostor, 37, 39
linearna algebra, 39
linearna regresija, 40
Llama, 12
Llama 2-70B, 46
Llama 3 70B, 10
Llama 3 8B, 10
logika prekidača, 150
lokalna razvojna okruženja, 143
Louvre, 39

Managed Streaming for Apache Kafka, 38
Markdown, 137
Mašine sa vektorima podrške (MVP), 113

medicinska otkrića, 93
mehanizmi ponovnih pokušaja, 102
mehanizmi za povratak, 243
Memorial Sloan Kettering Cancer Center, 38
Menadžer Procesa, 100
Menadžer procesa, 97
Merkur (element), 41
Merkur (planeta), 41
Merkur (rimski bog), 41
MessagePack, 235
Meta, 22
Metropolitan Museum of Art, 39
međumodalno generisanje, 20
Mistral, 23
  7B, 10
  7B Instruct, 15, 190
Mixtral
  8x22B, 10
  8x7B, 52
Mnoštvo radnika, 111, 154
modeli zasnovani na pretraživanju, 6
moderne aplikacije, 207
modularnost, 82
motivacione strategije, 198
mrežna povezanost, 210
Multimodalni
  jezički modeli, 19
  modeli, 18

Naivni Bajes, 113
naočare za proširenu realnost, 203
nenadgledano učenje, 4
neuralne mreže, 3, 6
New Relic, 234
nizovi, 122

objašnjivost, 240
obrada toka, 145
  logika, 146
obrada toka podataka, 139, 150
Obrasci poslovne integracije, 97
obrazovne aplikacije, 30
obuka na instrukcijama
  modeli obučeni na instrukcijama, 46, 48
odlučivanje
  slučajevi upotrebe, 124
  stabla, 206
Ograđivanje Odgovora, 190
Ograđivanje odgovora, 163
Ollama, 23
Olympia, 31, 58, 120, 133, 140, 155
Olympia-ina baza znanja, 85
online prodavci, 190
OpenAI, 3, 21, 36, 68
OpenRouter, 25, 26, 140, 233
OPT model, 22
optimističko zaključavanje, 102
orkestracija inteligentnog radnog toka, 213
osnovni modeli, 50
otklanjanje grešaka, 209
  i rešavanje problema, 230
označavanje pomoću markup jezika, 66

pametni telefoni, 203
parafraziranje, 49
paralelno izvršavanje, 232
parametar
    Broj parametara, 26
    efekti, 120
    opseg, 10
performanse
    kompromisi, 5
    optimizacija, 124, 182, 230
    problemi, 234
Perplexity (Provajder), 10
personalizacija, 174, 202, 207
    Personalizovani formulari, 186
    Personalizovani Mikrotekst, 191
personalizovane preporuke proizvoda, 85
pesimističko zaključavanje, 102
planiranje reagovanja u vanrednim
        situacijama, 30
podaci
    analiza, 32, 137
    integritet, 223
    postojanost, 102
    Preuzimanje podataka, 102
    priprema, 101
    privatnost, 200
    proces obrade, 223
    protok, 102
    Sinhronizacija podataka, 102
    Validacija podataka, 242
    zadaci obrade, 117
podaci za obuku, 39
podešavanje putem instrukcija, 9
Podrška kliničkom odlučivanju, 96
poruka okidač, 97
poslovna pravila, 206
poverenje korisnika, 201
povratna sprega
    Povratna sprega, 55
poziv alata, 142
poziv funkcije
    neuspeh, 125
praćenje
    i beleženje, 103, 229
    i upozoravanje, 211
    metrike, 230
praćenje ključnih metrika, 227
predviđanja, 5
Preporuke Proizvoda, 85
prevođenje, 15, 181
Prikupljanje medicinske istorije, 94
Primene u E-trgovini, 85
princip najmanje privilegije, 66
Prinudni odabir alata, 123
prirodni jezik
    Obrada prirodnog jezika (ONJ), 94
    Obrada prirodnog jezika (OPJ), 112
pristrasnost
    i pravičnost u AI, 240
pristupačnost, 201, 202
probabilistički modeli, 40
problemi upotrebljivosti, 201
Procena i stratifikacija simptoma, 94
proces destilacije, 71

Process Manager
Enterprise Integration, 213
Produktivnost, 176
progresivno otkrivanje, 192
promptovi
Destilacija promptova, 68, 72, 233
inženjering, 55
Prompt Object, 69
ulančavanje, 55, 66
Šablon Prompta, 190
Šablon promptova, 55
prompts
engineering, 60
propusnost, 25
Protocol Buffers, 235
protočni podaci, 141
provajderi hostinga modela otvorenog koda, 190
psihologija korisnika, 200
PyTorch, 22

Qwen2 70B, 10

Rails, 181
Railway Oriented Programming (ROP), 88
Raix, 214
biblioteka, 90
rangiratori, 33
razgovor
petlja, 148
transkript, 147
razvoj aplikacija, 205
razvojna okruženja, 243
razvojni okviri, 138
račun, 84
računarska nauka, 67
računarske nauke, 65
revizija i usklađenost, 230
revizorsko beleženje, 99
rezervne strategije, 102
rečnici, 122
RSpec, 237, 239, 242
Ruby, 86, 87, 105, 150, 242
Ruby on Rails, 1, 104, 213, 220
Rudall, Alex, 21
rukovanje izuzecima, 210, 212
rukovaoci toka, 139
Rust (Programski jezik), 108
Rust (programski jezik), 86
ručna intervencija, 212

sadržaj
Kategorizacija sadržaja, 104
Samozalečujući podaci, 152, 226
Scout, 234
server-sent events (SSE), 139
sintaksne greške, 123
sistemi za objavljivanje-pretplatu, 101
sistemi za odgovaranje na pitanja, 7
sistemska direktiva, 92, 120
skalabilnost, 207, 231
složeni zadaci, 136
softverska arhitektura, 2
sposobnosti donošenja odluka, 92
SQL injekcije, 65

strategije segmentacije i ciljanja, 180
Stratifikacija rizika, 95
Stripe, 121
Strukturirani IO, 190
strukturirani podaci, 125
strukturirano beleženje, 231
sumiranje, 48
suziti put, 36
sužavanje puta, 35

T5, 22
tableti, 203
tačke
    odlučivanja, 228
Temperatura, 50
teorija uma, 37
testiranje s kraja na kraj, 241, 242
Together.ai, 24
tokeni, 5, 11
tokenizacija, 11
Top-k uzorkovanje, 44
Top-p (nucleus) uzorkovanje, 44
tragedija zajedničkog dobra, 177
transformer arhitektura, 6
Trbuhozborac, 163

ulančavanje AI radnika, 103
ulaz
    upiti, 52
ulazni parametri, 120
ulazni podaci
    validacija, 237
Unicode-encodable language, 13
Universal ID, 236
uparivanje obrazaca, 141
upiti
    Destilacija upita, 42
    dizajn, 54, 63
    inženjering, 199
    inženjerstvo, 37, 41, 42, 52, 62
    usavršavanje, 63
upotreba alata, 115, 138
upravljanje saobraćajem, 30
upravljanje znanjem, 30
uska grla, 210
učenje bez primera, 54, 55
Učenje sa jednim primerom, 56
učenje sa malo primera, 57
    primena, 58

Veliki jezički model (LLM), 27, 103, 115, 116, 173, 184, 194
Veliki jezički model (VJM), 1, 3, 16, 62, 63, 66, 70, 72, 81, 112, 125, 131, 134, 136, 152, 155, 189, 216
verifikacija izlaznih podataka, 237
Verifikacija osiguranja, 94
većinsko glasanje, 109
veštačka inteligencija
    aplikacije, 138, 150
VI, 68, 120, 125
virtuelni asistenti, 31
vizuelni interfejs, 194
Višeagentni
    Rešavaoci problema, 29

višekoračni tok rada, 104
Vreme do prvog tokena (TTFT), 25
vreme obrade, 103

Wall, Larry, 3
Wisper, 87, 99, 140, 147
Wooley, Chad, 86

XML, 125

Yi-34B, 46

zadržavanje i rotacija zapisa, 231
Zaključivanje, 5
zatvoreno i otvoreno odgovaranje na pitanja, 48

Čišćenje teksta, 104
Čovek u petlji (HITL), 166
četbot aplikacija, 111
četbotovi za korisničku podršku, 31