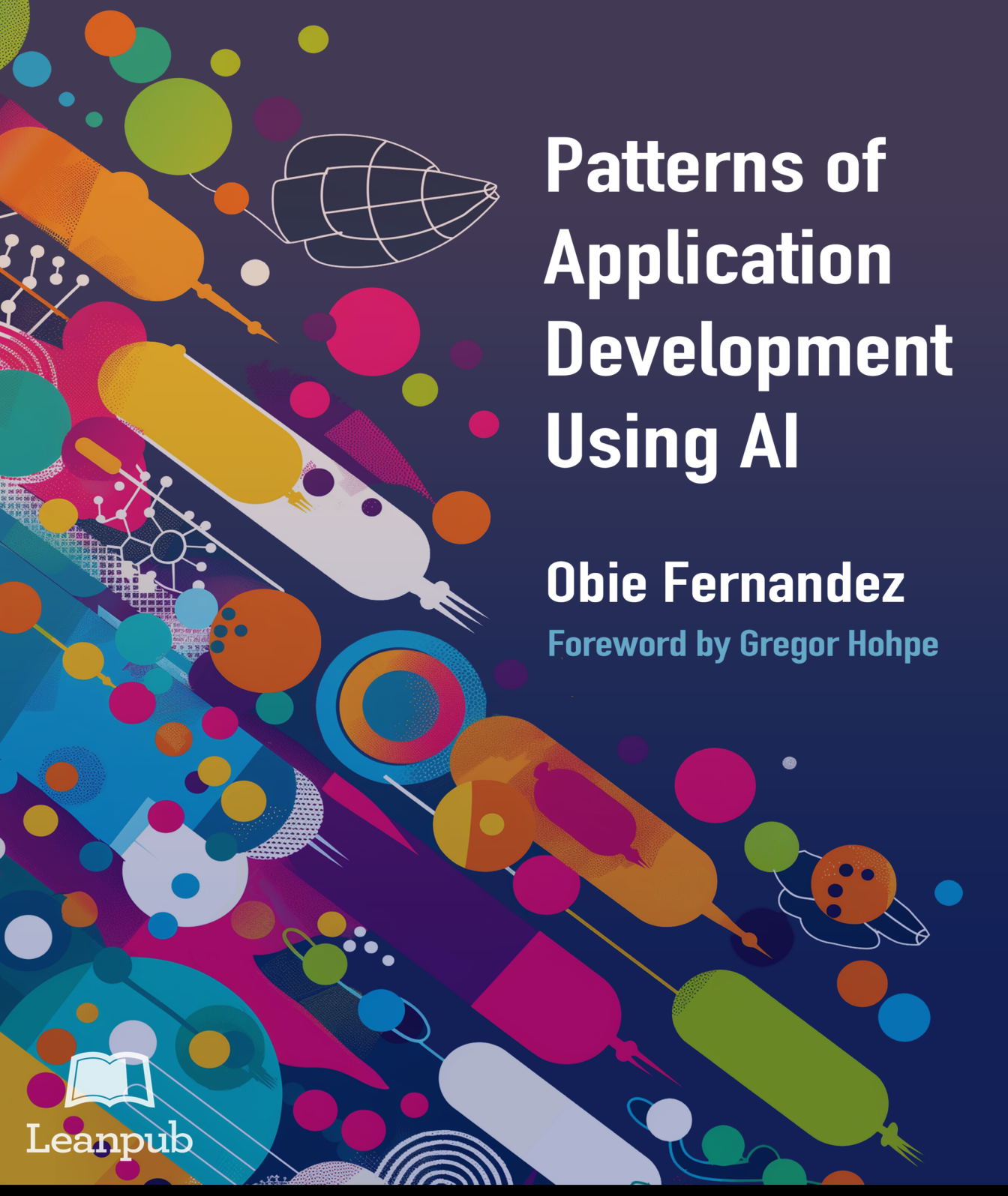


Hrvatsko Izdanje

Obrasci razvoja aplikacija korištenjem AI-ja (Hrvatsko Izdanje)

Obie Fernandez

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

Ova verzija je objavljena dana 2025-01-23

Tweet-ujte o ovoj knjizi!

Pomozite Obie Fernandez preporučujući ovu knjigu na Twitter!

Predloženi hashtag za ovu knjigu je #poaduai.

Saznajte što drugi kažu o ovoj knjizi klikom na link za pretragu ovog hashtag-a na Twitter-u:

#poaduai

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

Također od 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

Sadržaj

Predgovor Gregora Hohpea

Predgovor

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

O knjizi

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

O primjerima koda

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Što ne obrađujem

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Kome je ova knjiga namijenjena

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Izgradnja zajedničkog vokabulara

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Uključivanje

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Zahvale

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Što je s ilustracijama?

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

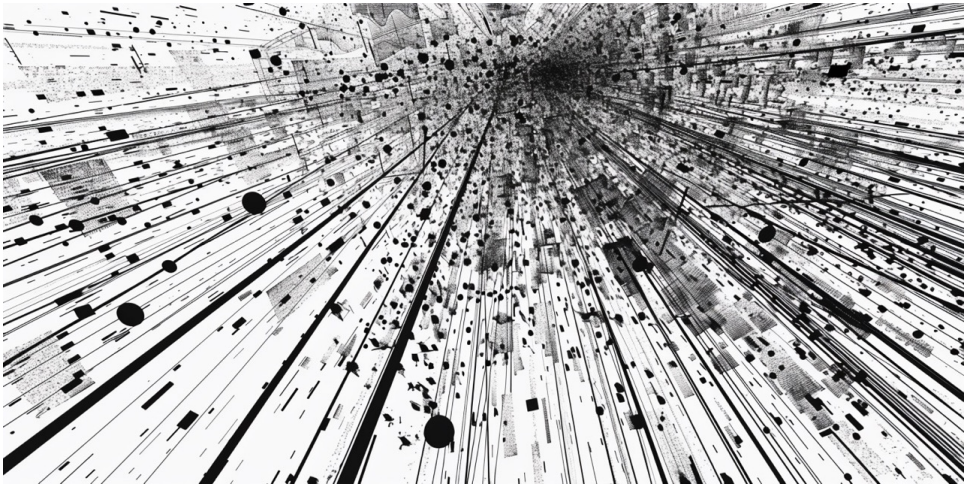
O Lean Publishingu

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

O Autoru

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Uvod



Ako ste željni početi integrirati velike jezične modele (LLM-ove) u svoje programske projekte, slobodno se odmah upustite u obrasce i primjere koda predstavljene u kasnijim poglavljima. Međutim, da biste u potpunosti cijenili snagu i potencijal ovih obrazaca, vrijedi odvojiti trenutak za razumijevanje šireg konteksta i kohezivnog pristupa koji predstavljaju.

Obrasci nisu samo zbirka izoliranih tehnika, već jedinstveni okvir za integraciju umjetne inteligencije u vaše aplikacije. Koristim Ruby on Rails, ali ovi bi obrasci trebali raditi u gotovo svakom drugom programskom okruženju. Oni se bave širokim spektrom problema, od upravljanja podacima i optimizacije performansi do korisničkog iskustva i sigurnosti, pružajući sveobuhvatan skup alata za unapređenje tradicionalnih programskih praksi mogućnostima umjetne inteligencije.

Svaka kategorija obrazaca bavi se specifičnim izazovom ili prilikom koja se pojavljuje pri ugradnji AI komponenti u vašu aplikaciju. Razumijevanjem odnosa i sinergija između

ovih obrazaca možete donositi informirane odluke o tome gdje i kako najučinkovitije primijeniti umjetnu inteligenciju.

Obrasci nikada nisu preskriptivna rješenja i ne treba ih tako tretirati. Oni su zamišljeni kao prilagodljivi građevni blokovi koje treba prilagoditi jedinstvenim zahtjevima i ograničenjima vaše vlastite jedinstvene aplikacije. Uspješna primjena ovih obrazaca (kao i bilo kojih drugih u području softvera) oslanja se na duboko razumijevanje problemske domene, korisničkih potreba i cjelokupne tehničke arhitekture vašeg projekta.

Razmišljanja o softverskoj arhitekturi

Počeo sam programirati 1980-ih i bio sam uključen u hakersku scenu, te nikada nisam izgubio svoj hakerski mentalitet, čak ni nakon što sam postao profesionalni programer. Od početka sam uvijek imao zdravu skepsu o tome kakvu vrijednost softverski arhitekti u svojim bjelokosnim kulama zapravo donose.

Jedan od razloga zašto sam osobno toliko uzbuđen zbog promjena koje donosi ovaj moćni novi val AI tehnologije je njegov utjecaj na ono što smatramo odlukama *softverske arhitekture*. On izaziva tradicionalne predodžbe o tome što čini "ispravan" način dizajniranja i implementacije naših softverskih projekata. Također dovodi u pitanje može li se arhitektura i dalje smatrati prvenstveno kao *dijelovi sustava koje je teško promijeniti*, s obzirom da AI poboljšanja čine lakšim nego ikad promjenu bilo kojeg dijela vašeg projekta, u bilo kojem trenutku.

Možda ulazimo u vrhunske godine "postmodernog" pristupa softverskom inženjerstvu. U ovom kontekstu, postmoderno se odnosi na fundamentalni odmak od tradicionalnih paradigmi, gdje su programeri bili odgovorni za pisanje i održavanje svake linije koda. Umjesto toga, prihvaća ideju delegiranja zadataka, poput manipulacije podacima, složenih algoritama, pa čak i čitavih dijelova aplikacijske logike, vanjskim bibliotekama i API-jima. Ovaj postmoderni pomak predstavlja značajan odmak od konvencionalne mudrosti izgradnje aplikacija od temelja, i izaziva programere da preispitaju svoju ulogu u razvojnom procesu.

Oduvijek sam vjerovao da dobri programeri pišu samo kod koji je apsolutno nužno napisati, temeljeno na učenjima Larryja Walla i drugih hakerskih luminara poput njega. Minimiziranjem količine napisanog koda, možemo se kretati brže, smanjiti površinu za bugove, pojednostaviti održavanje i poboljšati ukupnu pouzdanost naših aplikacija. Manje koda omogućuje nam da se fokusiramo na osnovnu poslovnu logiku i korisničko iskustvo, dok ostali posao delegiramo drugim servisima.

Sada kada AI sustavi mogu obavljati zadatke koji su prije bili isključivo domena ljudski napisanog koda, trebali bismo moći biti još produktivniji i agilniji, s većim fokusom nego ikad na stvaranju poslovne vrijednosti i korisničkog iskustva.

Naravno, postoje kompromisi kod delegiranja velikih dijelova vašeg projekta AI sustavima, poput potencijalnog gubitka kontrole i potrebe za robusnim mehanizmima praćenja i povratnih informacija. Zato to zahtijeva novi set vještina i znanja, uključujući barem osnovno razumijevanje kako AI funkcionira.

Što je veliki jezični model?

Veliki jezični modeli (LLM-ovi) su vrsta modela umjetne inteligencije koji su privukli značajnu pažnju posljednjih godina, još od lansiranja GPT-3 od strane OpenAI-ja 2020. godine. LLM-ovi su dizajnirani za obradu, razumijevanje i generiranje ljudskog jezika s izvanrednom preciznošću i točnošću. U ovom odjeljku, kratko ćemo pogledati kako LLM-ovi funkcioniraju i zašto su pogodni za izgradnju inteligentnih komponenti sustava.

U svojoj srži, LLM-ovi se temelje na algoritmima dubokog učenja, specifično neuronskim mrežama. Ove mreže sastoje se od međusobno povezanih čvorova, ili neurona, koji obrađuju i prenose informacije. Arhitektura izbora za LLM-ove često je Transformer model, koji se pokazao vrlo učinkovitim u obradi sekvencijalnih podataka poput teksta.

Transformerski modeli temelje se na mehanizmu pažnje i prvenstveno se koriste za zadatke koji uključuju sekvencijalne podatke, poput obrade prirodnog jezika. Transformeri obrađuju ulazne podatke odjednom umjesto sekvencijalno, što im

omogućuje učinkovitije hvatanje zavisnosti na većim udaljenostima. Imaju slojeve mehanizama pažnje koji pomažu modelu da se usredotoči na različite dijelove ulaznih podataka kako bi razumio kontekst i odnose.

Proces treniranja velikih jezičnih modela uključuje izlaganje modela ogromnim količinama tekstualnih podataka, kao što su knjige, članci, web stranice i repozitoriji koda. Tijekom treniranja, model uči prepoznavati uzorke, odnose i strukture unutar teksta. Hvata statističke značajke jezika, poput gramatičkih pravila, povezanosti riječi i kontekstualnih značenja.

Jedna od ključnih tehnika koja se koristi u treniranju velikih jezičnih modela je nenadgledano učenje. To znači da model uči iz podataka bez izričitog označavanja ili vođenja. Sam otkriva uzorke i reprezentacije analizirajući zajedničko pojavljivanje riječi i fraza u podacima za treniranje. To omogućuje velikim jezičnim modelima da razviju duboko razumijevanje jezika i njegovih složenosti.

Još jedan važan aspekt velikih jezičnih modela je njihova sposobnost rukovanja *kontekstom*. Prilikom obrade teksta, veliki jezični modeli ne uzimaju u obzir samo pojedinačne riječi, već i okolni kontekst. Uzimaju u obzir prethodne riječi, rečenice, pa čak i odlomke kako bi razumjeli značenje i namjeru teksta. Ovo kontekstualno razumijevanje omogućuje velikim jezičnim modelima generiranje koherentnih i relevantnih odgovora. Jedan od glavnih načina na koji procjenjujemo sposobnosti određenog jezičnog modela jest razmatranje veličine konteksta koji mogu uzeti u obzir pri generiranju odgovora.

Nakon treniranja, veliki jezični modeli mogu se koristiti za širok raspon jezičnih zadataka. Mogu generirati tekst nalik ljudskom, odgovarati na pitanja, sažimati dokumente, prevoditi jezike, pa čak i pisati kod. Svestranost velikih jezičnih modela čini ih vrijednima za izgradnju inteligentnih sistemskih komponenti koje mogu komunicirati s korisnicima, obrađivati i analizirati tekstualne podatke te generirati smislene izlazne podatke.

Ugrađivanjem velikih jezičnih modela u arhitekturu aplikacije možete stvoriti AI

komponente koje razumiju i obrađuju korisnički unos, generiraju dinamički sadržaj i pružaju inteligentne preporuke ili akcije. No rad s velikim jezičnim modelima zahtijeva pažljivo razmatranje resursnih zahtjeva i kompromisa u performansama. Veliki jezični modeli računski su zahtjevni i mogu trebati značajnu procesorsku snagu i memoriju (drugim riječima, novac) za rad. Većina nas morat će procijeniti troškovne implikacije integracije velikih jezičnih modela u naše aplikacije i djelovati u skladu s tim.

Razumijevanje zaključivanja

Zaključivanje se odnosi na proces kojim model generira predviđanja ili izlazne podatke na temelju novih, neviđenih podataka. To je faza u kojoj se trenirani model koristi za donošenje odluka ili generiranje teksta, slika ili drugog sadržaja kao odgovor na korisničke unose.

Tijekom faze treniranja, AI model uči iz velikog skupa podataka prilagođavanjem svojih parametara kako bi minimizirao pogreške u svojim predviđanjima. Nakon treniranja, model može primijeniti naučeno na nove podatke. Zaključivanje je način na koji model koristi svoje naučene uzorke i znanje za generiranje izlaznih podataka.

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

Za razliku od načina na koji vi i ja razmišljamo, “razmišljanje” AI modela putem zaključivanja događa se u jednoj operaciji bez stanja. To jest, njegovo razmišljanje ograničeno je na proces generiranja. Doslovno mora razmišljati naglas, kao da sam vas pitao pitanje i prihvaćao odgovor samo u stilu “toka svijesti”.

Veliki jezični modeli dolaze u mnogim veličinama i okusima

Iako su praktički svi popularni veliki jezični modeli temeljeni na istoj osnovnoj transformerskoj arhitekturi i trenirani na ogromnim tekstualnim skupovima podataka, dolaze u različitim veličinama i fino su podešeni za različite svrhe. Veličina velikog jezičnog modela, mjerena brojem parametara u njegovoj neuronskoj mreži, ima velik utjecaj na njegove sposobnosti. Veći modeli s više parametara, poput GPT-4, za koji se govori da ima 1 do 2 bilijuna parametara, općenito su obrazovaniji i sposobniji od manjih modela. Međutim, veći modeli također zahtijevaju mnogo više računalne snage za rad, što se prevodi u veće troškove kada ih koristite putem API poziva.

Kako bi veliki jezični modeli bili praktičniji i prilagođeniji specifičnim slučajevima uporabe, osnovni modeli često se fino podešavaju na ciljanijim skupovima podataka. Na primjer, veliki jezični model može biti treniran na velikom korpusu dijaloga kako bi se specijalizirao za konverzacijsku umjetnu inteligenciju. Drugi su trenirani na kodu kako bi im se usadilo programersko znanje. Postoje čak i modeli koji su posebno trenirani za interakcije s korisnicima u stilu igranja uloga!

Modeli temeljeni na dohvaćanju nasuprot generativnih modela

U svijetu velikih jezičnih modela (LLM-ova), postoje dva glavna pristupa generiranju odgovora: modeli temeljeni na dohvaćanju i generativni modeli. Svaki pristup ima svoje prednosti i nedostatke, a razumijevanje razlika između njih može vam pomoći odabrati pravi model za vaš specifični slučaj uporabe.

Modeli temeljeni na dohvaćanju

Modeli temeljeni na dohvaćanju, poznati i kao modeli za dohvat informacija, generiraju odgovore pretraživanjem velike baze postojećeg teksta i odabirom najrelevantnijih

odlomaka na temelju ulaznog upita. Ovi modeli ne generiraju novi tekst ispočetka, već spajaju dijelove iz baze podataka kako bi formirali koherentan odgovor.

Jedna od glavnih prednosti modela temeljenih na dohvaćanju je njihova sposobnost pružanja činjenično točnih i ažurnih informacija. Budući da se oslanjaju na bazu kuriranog teksta, mogu izvući relevantne informacije iz pouzdanih izvora i predstaviti ih korisniku. To ih čini pogodnima za aplikacije koje zahtijevaju precizne, činjenične odgovore, poput sustava za odgovaranje na pitanja ili baza znanja.

Međutim, modeli temeljeni na dohvaćanju imaju određena ograničenja. Dobri su onoliko koliko je dobra baza podataka koju pretražuju, tako da kvaliteta i pokrivenost baze podataka izravno utječu na performanse modela. Također, ovi modeli mogu imati poteškoća s generiranjem koherentnih i prirodno zvučećih odgovora, jer su ograničeni na tekst dostupan u bazi podataka.

U ovoj knjizi ne obrađujemo uporabu čistih modela za dohvaćanje.

Generativni modeli

Generativni modeli, s druge strane, stvaraju novi tekst ispočetka na temelju obrazaca i odnosa koje su naučili tijekom treninga. Ovi modeli koriste svoje razumijevanje jezika za generiranje novih odgovora koji su prilagođeni ulaznom upitu.

Glavna snaga generativnih modela je njihova sposobnost proizvodnje kreativnog, koherentnog i kontekstualno relevantnog teksta. Mogu voditi otvorene razgovore, generirati priče, pa čak i pisati kod. To ih čini idealnima za aplikacije koje zahtijevaju otvorenije i dinamičnije interakcije, poput chatbotova, stvaranja sadržaja i pomoćnika za kreativno pisanje.

Međutim, generativni modeli ponekad mogu proizvesti nedosljedne ili činjenično netočne informacije, jer se oslanjaju na obrasce naučene tijekom treninga, a ne na kuriranu bazu činjenica. Također mogu biti skloniji pristranostima i halucinacijama, generirajući tekst koji je uvjerljiv, ali ne nužno istinit.

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

Hibridni modeli

Nekoliko komercijalno dostupnih LLM-ova kombinira pristupe dohvaćanja i generiranja u hibridnom modelu. Ovi modeli koriste tehnike dohvaćanja za pronalaženje relevantnih informacija iz baze podataka, a zatim koriste generativne tehnike za sintezu tih informacija u koherentan odgovor.

Hibridni modeli nastoje kombinirati činjeničnu točnost modela temeljenih na dohvaćanju s mogućnostima generiranja prirodnog jezika generativnih modela. Mogu pružiti pouzdanije i ažurnije informacije dok istovremeno zadržavaju sposobnost vođenja otvorenih razgovora.

Pri odabiru između modela temeljenih na dohvaćanju i generativnih modela, trebali biste uzeti u obzir specifične zahtjeve vaše aplikacije. Ako je primarni cilj pružanje točnih, činjeničnih informacija, model temeljen na dohvaćanju može biti najbolji izbor. Ako aplikacija zahtijeva otvorenije i kreativnije interakcije, generativni model može biti prikladniji. Hibridni modeli nude ravnotežu između dva pristupa i mogu biti dobar izbor za aplikacije koje zahtijevaju i činjeničnu točnost i generiranje prirodnog jezika.

U konačnici, izbor između modela temeljenih na dohvaćanju i generativnih modela ovisi o specifičnom slučaju uporabe i kompromisima između točnosti, kreativnosti i fleksibilnosti. Razumijevanjem prednosti i ograničenja svakog pristupa možete donositi informirane odluke.

Podešavanje putem instrukcija

Podešavanje putem instrukcija je podskup konverzacijskog treninga koji koristi ljudski napisane upite i odgovore kako bi LLM-ovi bolje razumjeli i slijedili upute. Evo nekoliko jednostavnih primjera koji ilustriraju kako podešavanje putem instrukcija na ljudski napisanim upitima i odgovorima poboljšava sposobnost LLM-ova da razumiju i slijede upute:

Primjer 1: Korisnička podrška

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

Ljudski napisan upit: "Želio/la bih zatražiti povrat novca za svoju nedavnu kupnju. Proizvod nije ispunio moja očekivanja."

Ljudski napisan odgovor: "Žao mi je što proizvod nije ispunio vaša očekivanja. Za obradu povrata novca, molimo vas da navedete broj narudžbe i razlog za povrat. Pregledat ćemo vaš zahtjev i javiti vam se unutar 2-3 radna dana."

Primjer 2: Obrazovna pomoć

Instrukcija: "Objasni koncept fotosinteze desetogodišnjaku."

Ljudski napisani upit: "Možeš li objasniti kako biljke stvaraju svoju hranu?"

Ljudski napisan odgovor: "Naravno! Biljke stvaraju svoju hranu kroz proces koji se zove fotosinteza. Koriste sunčevu svjetlost, vodu i ugljikov dioksid za stvaranje hrane i kisika. Lišće biljke ima poseban dio koji se zove klorofil koji im pomaže uhvatiti sunčevu svjetlost."

Primjer 3: Osobni asistent

Uputa: "Zakaži sastanak za sljedeći utorak u 15 sati."

> Ljudski napisani upit: “Možeš li postaviti sastanak za utorak u 15 sati?”
>
> Ljudski napisan odgovor: “Naravno, zakazao sam vam sastanak za sljedeći utorak u 15 sati. Treba li vam još nešto?”

Rezultat je raznolik ekosustav velikih jezičnih modela različitih veličina i s različitim specijalizacijama. Manji modeli u rasponu od 1-7 milijardi parametara pružaju dobre opće jezične sposobnosti uz veću učinkovitost izvođenja.

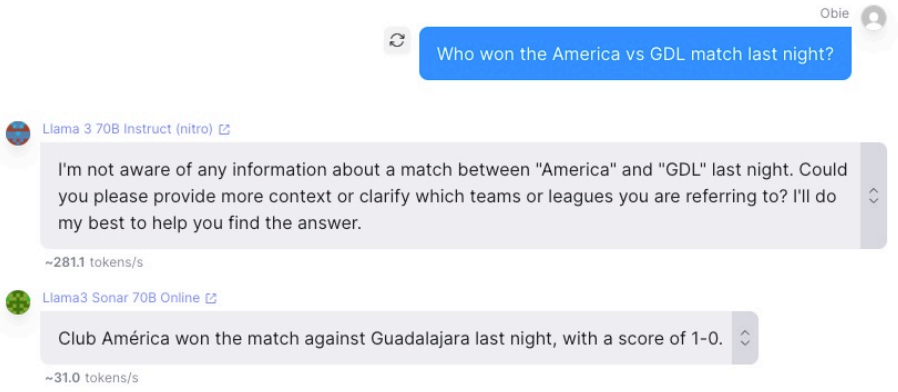
- Mistral 7B
- Llama 3 8B
- Gemma 7B

Modeli srednje veličine s oko 30-70 milijardi parametara nude jače sposobnosti zaključivanja i praćenja uputa.

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

Pri odabiru velikog jezičnog modela za ugradnju u aplikaciju, morate uravnotežiti mogućnosti modela s praktičnim čimbenicima poput troškova, latencije, duljine konteksta i filtriranja sadržaja. Manji modeli podešeni za upute često su najbolji izbor za jednostavnije jezične zadatke, dok najveći modeli mogu biti potrebni za složeno zaključivanje ili analizu. Podaci za treniranje modela također su važan čimbenik, jer određuju datum završetka znanja modela.

Određeni modeli, poput nekih od Perplexityja, povezani su s izvorima informacija u stvarnom vremenu, tako da efektivno nemaju datum završetka. Kada im postavite pitanja, mogu samostalno odlučiti pretraživati web i dohvatiti proizvoljne web stranice kako bi generirali odgovor.

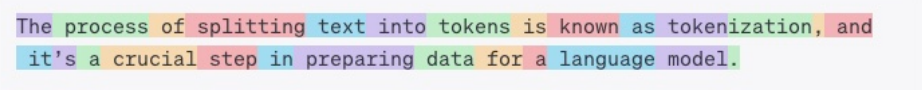


Slika 1. Llama3 s i bez online pristupa

U konačnici, ne postoji univerzalni veliki jezični model. Razumijevanje varijacija u veličini modela, arhitekturi i treniranju ključno je za odabir pravog modela za određeni slučaj uporabe. Eksperimentiranje s različitim modelima jedini je praktični način da se otkrije koji pružaju najbolje performanse za zadani zadatak.

Tokenizacija: Razbijanje teksta na dijelove

Prije nego što veliki jezični model može obraditi tekst, taj tekst treba biti razbijen na manje jedinice koje se zovu *tokeni*. Tokeni mogu biti pojedinačne riječi, dijelovi riječi ili čak pojedinačni znakovi. Proces razbijanja teksta na tokene naziva se tokenizacija, i to je ključni korak u pripremi podataka za jezični model.



Slika 2. Ova rečenica sadrži 27 tokena

Različiti veliki jezični modeli koriste različite strategije tokenizacije, što može značajno utjecati na performanse i mogućnosti modela. Neki uobičajeni tokenizatori koje koriste

veliki jezični modeli uključuju:

- **GPT (Kodiranje parova bajtova)**: GPT tokenizatori koriste tehniku zvanu kodiranje parova bajtova (BPE) za razbijanje teksta na podriječne jedinice. BPE iterativno spaja najčešće parove bajtova u tekstualnom korpusu, formirajući vokabular podriječnih tokena. To omogućuje tokenizatoru da obrađuje rijetke i nove riječi razbijajući ih na češće podriječne dijelove. GPT tokenizatore koriste modeli poput GPT-3 i GPT-4.

- **Llama (SentencePiece)**: Llama tokenizatori koriste SentencePiece biblioteku, koja je nenadziran tokenizator i detokenizator teksta. SentencePiece tretira ulazni tekst kao niz Unicode znakova i uči vokabular podriječi na temelju korpusa za treniranje. Može obrađivati bilo koji jezik koji se može kodirati u Unicodeu, što ga čini pogodnim za višejezične modele. Llama tokenizatore koriste modeli poput Metinog Llama i Alpaca.

- **SentencePiece (Unigram)**: SentencePiece tokenizatori također mogu koristiti drugačiji algoritam nazvan Unigram, koji se temelji na tehnici regulacije podriječi. Unigram tokenizacija određuje optimalan vokabular podriječi na temelju unigram jezičnog modela, koji dodjeljuje vjerojatnosti pojedinačnim jedinicama podriječi. Ovaj pristup može proizvesti semantički značajnije podriječi u usporedbi s BPE-om. SentencePiece s Unigramom koriste modeli poput Googleovog T5 i BERT-a.

- **Google Gemini (Višemodalna Tokenizacija)**: Google Gemini koristi shemu tokenizacije dizajniranu za rukovanje različitim vrstama podataka, uključujući tekst, slike, zvuk, videozapise i kod. Ova višemodalna sposobnost omogućuje Geminiju obradu i integraciju različitih oblika informacija. Posebno je značajno da Google Gemini 1.5 Pro ima kontekstni prozor koji može obraditi milijune tokena, što je mnogo više od prethodnih modela. Ovaj opsežni kontekstni prozor omogućuje modelu obradu većeg konteksta, što potencijalno dovodi do preciznijih

odgovora. Međutim, važno je napomenuti da je Geminijeva shema tokenizacije mnogo bliža jednom tokenu po znaku nego kod drugih modela. To znači da stvarni trošak korištenja Gemini modela može biti značajno veći nego što se očekuje ako ste navikli koristiti modele poput GPT-a, jer se Googleovo određivanje cijena temelji na znakovima, a ne na tokenima.

Izbor tokenizatora utječe 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 raspon riječi i fraza te čak postane višemodalan (sposoban razumjeti i generirati više od samog teksta), ali također povećava memorijske zahtjeve modela i računalnu složenost.
- **Rukovanje rijetkim i nepoznatim riječima**: Tokenizatori koji koriste jedinice podriječi, poput BPE-a i SentencePiece-a, mogu razbiti rijetke i nepoznate riječi u češće dijelove podriječi. To omogućuje modelu da donosi obrazovane pretpostavke o značenju riječi koje prije nije vidio, na temelju podriječi koje sadrže.
- **Višejezična podrška**: Tokenizatori poput SentencePiece-a, koji mogu obrađivati bilo koji jezik koji se može kodirati u Unicode-u, pogodni su za višejezične modele koji trebaju obrađivati tekst na više jezika.

Pri odabiru LLM-a za određenu primjenu, važno je razmotriti tokenizator koji koristi i koliko dobro se usklađuje s specifičnim potrebama obrade jezika za zadani zadatak. Tokenizator može imati značajan utjecaj na sposobnost modela da obrađuje terminologiju specifičnu za domenu, rijetke riječi i višejezični tekst.

Veličina konteksta: Koliko informacija jezični model može koristiti tijekom zaključivanja?

Kada se govori o jezičnim modelima, veličina konteksta odnosi se na količinu teksta koju model može razmotriti pri obradi ili generiranju svojih odgovora. To je u osnovi

mjera koliko informacija model može "zapamtiti" i koristiti za oblikovanje svojih izlaza (izraženo u tokenima). Veličina konteksta jezičnog modela može značajno utjecati na njegove mogućnosti i vrste zadataka koje može učinkovito izvršavati.

Što je veličina konteksta?

U tehničkom smislu, veličina konteksta određena je brojem tokena (riječi ili dijelova riječi) koje jezični model može obraditi u jednom ulaznom nizu. To se često naziva "rasponom pažnje" ili "kontekstnim prozorom" modela. Što je veća veličina konteksta, više teksta model može istovremeno razmotriti pri generiranju odgovora ili izvršavanju zadatka.

Različiti jezični modeli imaju različite veličine konteksta, u rasponu od nekoliko stotina tokena do milijuna tokena. Za usporedbu, tipičan odlomak teksta može sadržavati oko 100-150 tokena, dok cijela knjiga može sadržavati desetke ili stotine tisuća tokena.

> Postoje čak i istraživanja o učinkovitim metodama za skaliranje Transformer-baziranih velikih jezičnih modela (LLM) na beskonačno duge unose s ograničenom memorijom i računanjem.

Zašto je veličina konteksta važna?

Veličina konteksta jezičnog modela značajno utječe na njegovu sposobnost razumijevanja i generiranja koherentnog, kontekstualno relevantnog teksta. Evo nekoliko ključnih razloga zašto je veličina konteksta važna:

1. **Razumijevanje dugačkih sadržaja**: Modeli s većim kontekstom mogu bolje razumjeti i analizirati duže tekstove, poput članaka, izvještaja, pa čak i cijelih knjiga. Ovo je ključno za zadatke poput sažimanja dokumenata, odgovaranja na pitanja i analize sadržaja.

2. **Održavanje koherentnosti**: Veći kontekstni prozor omogućuje modelu održavanje koherentnosti i dosljednosti kroz duže dijelove izlaznog teksta. Ovo je važno za zadatke poput generiranja priča, sustava za dijalog i stvaranja sadržaja, gdje je održavanje dosljedne naracije ili teme ključno. Također je apsolutno presudno kada se VJM-ovi koriste za generiranje ili transformaciju strukturiranih podataka.
3. **Hvatanje dalekosežnih zavisnosti**: Neki jezični zadaci zahtijevaju razumijevanje odnosa između riječi ili fraza koje su međusobno udaljene u tekstu. Modeli s većim kontekstom bolje su opremljeni za hvatanje tih dalekosežnih zavisnosti, što može biti važno za zadatke poput analize sentimenta, prevođenja i jezičnog razumijevanja.
4. **Rukovanje složenim uputama**: U aplikacijama gdje se jezični modeli koriste za praćenje složenih, višekoračnih uputa, veća veličina konteksta omogućuje modelu da razmotri cijeli set uputa pri generiranju odgovora, umjesto samo nekoliko najnovijih riječi.

Primjeri jezičnih modela s različitim veličinama konteksta

Evo nekoliko primjera jezičnih modela s 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 vidjeti, 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 s većim veličinama konteksta (npr. “32k” verzije), koje mogu obraditi još duže ulazne nizove. A trenutno (travanj 2024.) Google Gemini Pro se može pohvaliti s gotovo 3 milijuna tokena!

Vrijedi napomenuti da veličina konteksta može varirati ovisno o specifičnoj implementaciji i verziji određenog modela. Na primjer, izvorni OpenAI GPT-4 model ima veličinu konteksta od 8.191 tokena, dok kasnije GPT-4 varijante poput Turbo i 4o imaju znatno veću veličinu konteksta od 128.000 tokena.

> Sam Altman je usporedio trenutna kontekstna ograničenja s kilobajtima radne memorije s kojima su se programeri osobnih računala morali nositi u 80-ima, i rekao da ćemo u bliskoj budućnosti moći uklopiti “sve svoje osobne podatke” u kontekst velikog jezičnog modela.

Odabir prave veličine konteksta

Pri odabiru jezičnog modela za određenu aplikaciju, važno je razmotriti zahtjeve zadatka za veličinom konteksta. Za zadatke koji uključuju kratke, izolirane dijelove teksta, poput analize sentimenta ili jednostavnog odgovaranja na pitanja, manja veličina konteksta može biti dovoljna. Međutim, za zadatke koji zahtijevaju razumijevanje i generiranje dužih, složenijih tekstova, vjerojatno će biti potrebna veća veličina konteksta.

Vrijedi napomenuti da veće veličine konteksta često dolaze s povećanim računalnim troškovima i sporijim vremenom obrade, jer model mora razmotriti više informacija pri generiranju odgovora. Stoga morate postići ravnotežu između veličine konteksta i performansi pri odabiru jezičnog modela za vašu aplikaciju.

> Zašto jednostavno ne odabrati model s najvećom veličinom konteksta i napuniti ga s što više informacija? Pa, osim faktora performansi, druga glavna stavka je trošak. U ožujku 2024. jedan *jedini* ciklus upita i odgovora koristeći Google Gemini Pro 1.5 s punim kontekstom koštat će vas gotovo 8 dolara (USD). Ako imate slučaj uporabe koji opravdava taj trošak, samo naprijed! Ali za većinu aplikacija, to je jednostavno preskupo za nekoliko redova veličine.

Pronalaženje Igle u Plastu Sijena

Koncept pronalaženja igle u plastu sijena dugo je bio metafora za izazove dohvaćanja podataka u velikim skupovima podataka. U području velikih jezičnih modela, malo prilagođavamo ovu analogiju. Zamislite da ne tražimo samo jednu činjenicu zakopanu unutar ogromnog teksta (poput cijele antologije eseja Paula Grahama), već više činjenica rasutih posvuda. Ovaj scenarij više nalikuje pronalaženju nekoliko igala u prostranom polju, a ne samo u jednom plastu sijena. Evo začkoljice: ne samo da moramo locirati te igle, već ih moramo i splesti u povezanu nit.

Kada se suoče sa zadatkom dohvaćanja i zaključivanja o višestrukim činjenicama ugrađenim u dugačke kontekste, veliki jezični modeli suočavaju se s dvostrukim izazovom. Prvo, postoji izravno pitanje točnosti dohvaćanja—ona prirodno opada kako se broj činjenica povećava. To je očekivano; naposlijetku, praćenje višestrukih detalja kroz opsežan tekst opterećuje čak i najsofisticiranije modele.

Drugo, i možda kritičnije, je izazov zaključivanja s tim činjenicama. Jedno je izdvojiti činjenice; sasvim drugo je sintetizirati ih u koherentnu priču ili odgovor. Tu dolazi pravi test. Učinkovitost velikih jezičnih modela u zadacima zaključivanja ima tendenciju degradirati više nego u jednostavnim zadacima dohvaćanja. Ta degradacija nije samo pitanje količine; radi se o složenom plesu konteksta, relevantnosti i zaključivanja.

Zašto se to događa? Pa, razmotrite dinamiku pamćenja i pažnje u ljudskoj kogniciji, koja

se do određene mjere odražava u velikim jezičnim modelima. Prilikom obrade velikih količina informacija, veliki jezični modeli, poput ljudi, mogu izgubiti trag ranijih detalja dok upijaju nove. Ovo je posebno točno kod modela koji nisu eksplicitno dizajnirani da automatski daju prioritet ili se vraćaju na ranije segmente teksta.

Štoviše, sposobnost velikog jezičnog modela da splete ove dohvaćene činjenice u koherentan odgovor slična je izgradnji narativa. To ne zahtijeva samo dohvaćanje informacija već duboko razumijevanje i kontekstualno postavljanje, što ostaje težak izazov za trenutnu umjetnu inteligenciju.

Dakle, što to znači za nas kao razvijatelje i integratore ovih tehnologija? Moramo biti izrazito svjesni ovih ograničenja pri dizajniranju sustava koji se oslanjaju na velike jezične modele za rukovanje složenim zadacima s dugačkim tekstovima. Razumijevanje da se učinkovitost može degradirati pod određenim uvjetima pomaže nam postaviti realna očekivanja i razviti bolje rezervne mehanizme ili dopunske strategije.

Modaliteti: Više od Teksta

Dok je većina današnjih jezičnih modela usredotočena na obradu i generiranje teksta, postoji rastući trend prema multimodalnim modelima koji mogu prirodno primati i proizvoditi više vrsta podataka, poput slika, zvuka i videa. Ovi multimodalni modeli otvaraju nove mogućnosti za aplikacije pokretane umjetnom inteligencijom koje mogu razumjeti i generirati sadržaj kroz različite modalitete.

Što su Modaliteti?

U kontekstu jezičnih modela, modaliteti se odnose na različite vrste podataka koje model može obrađivati i generirati. Najčešći modalitet je tekst, koji uključuje pisani jezik u raznim oblicima poput knjiga, članaka, web stranica i objava na društvenim mrežama. Međutim, postoji nekoliko drugih modaliteta koji se sve više uključuju u jezične modele:

- **Slike**: Vizualni podaci poput fotografija, ilustracija i dijagrama.

- **Audio**: Zvučni podaci poput govora, glazbe i zvukova iz okoline.
- **Video**: Pokretni vizualni podaci, često praćeni zvukom, poput video isječaka i filmova.

Svaki modalitet predstavlja jedinstvene izazove i mogućnosti za jezične modele. Na primjer, slike zahtijevaju da model razumije vizualne koncepte i odnose, dok audio zahtijeva da model obrađuje i generira govor i druge zvukove.

Multimodalni Jezični Modeli

Multimodalni jezični modeli dizajnirani su za rukovanje višestrukim modalitetima unutar jednog modela. Ovi modeli tipično imaju specijalizirane komponente ili slojeve koji mogu razumjeti ulazne podatke i generirati izlazne podatke u različitim modalitetima. Neki značajni primjeri multimodalnih jezičnih modela uključuju:

- **OpenAI-jev GPT-4o**: GPT-4o je veliki jezični model koji prirodno razumije i obrađuje govorni audio uz tekst. Ova sposobnost omogućuje GPT-4o izvođenje zadataka poput transkripcije govornog jezika, generiranja teksta iz audio ulaza i pružanja odgovora na temelju govornih upita.
- **OpenAI-jev GPT-4 s vizualnim unosom**: GPT-4 je veliki jezični model koji može obrađivati i tekst i slike. Kada mu se daje slika kao ulaz, GPT-4 može analizirati sadržaj slike i generirati tekst koji opisuje ili odgovara na vizualne informacije.
- **Googleov Gemini**: Gemini je multimodalni model koji može rukovati tekstom, slikama i videom. Koristi jedinstvenu arhitekturu koja omogućuje križno-modalno razumijevanje i generiranje, omogućujući zadatke poput opisivanja slika, sažimanja videa i odgovaranja na vizualna pitanja.

- **DALL-E i Stable Diffusion**: Iako nisu jezični modeli u tradicionalnom smislu, ovi modeli pokazuju snagu višemodalne umjetne inteligencije generiranjem slika iz tekstualnih opisa. Oni demonstriraju potencijal modela koji mogu prevoditi između različitih modaliteta.

Prednosti i primjene višemodalnih modela

Višemodalni jezični modeli nude nekoliko prednosti i omogućuju širok raspon primjena, uključujući:

- **Poboljšano razumijevanje**: Obradom informacija iz više modaliteta, ovi modeli mogu steći sveobuhvatnije razumijevanje svijeta, slično načinu na koji ljudi uče iz različitih osjetilnih inputa.
- **Križno-modalno generiranje**: Višemodalni modeli mogu generirati sadržaj u jednom modalitetu na temelju unosa iz drugog, poput stvaranja slike iz tekstualnog opisa ili generiranja video sažetka iz pisanog članka.
- **Pristupačnost**: Višemodalni modeli mogu učiniti informacije pristupačnijima prevođenjem između modaliteta, kao što je generiranje tekstualnih opisa slika za osobe s oštećenjem vida ili stvaranje audio verzija pisanog sadržaja.
- **Kreativne primjene**: Višemodalni modeli mogu se koristiti za kreativne zadatke poput generiranja umjetnosti, glazbe ili videa na temelju tekstualnih uputa, otvarajući nove mogućnosti za umjetnike i stvaratelje sadržaja.

Kako višemodalni jezični modeli nastavljaju napredovati, vjerojatno će igrati sve važniju ulogu u razvoju aplikacija temeljenih na umjetnoj inteligenciji koje mogu razumjeti i generirati sadržaj kroz više modaliteta. To će omogućiti prirodnije i intuitivnije interakcije između ljudi i AI sustava, kao i otključati nove mogućnosti za kreativno izražavanje i širenje znanja.

Ekosustavi pružatelja usluga

Kada je riječ o integraciji velikih jezičnih modela (LLM-ova) u aplikacije, imate rastući raspon opcija za odabir. Svaki veliki pružatelj LLM-ova, poput OpenAI, Anthropic, Google i Cohere, nudi svoj vlastiti ekosustav modela, API-ja i alata. Odabir pravog pružatelja uključuje razmatranje različitih faktora, uključujući cijene, performanse, filtriranje sadržaja, privatnost podataka i opcije prilagodbe.

OpenAI

OpenAI je jedan od najpoznatijih pružatelja LLM-ova, čija je GPT serija (GPT-3, GPT-4) široko korištena u različitim aplikacijama. OpenAI nudi jednostavan API koji vam omogućuje laku integraciju njihovih modela u aplikacije. Nude niz modela s različitim mogućnostima i cjenovnim razredima, od početnog Ada modela do moćnog Davinci modela.

OpenAI-jev ekosustav također uključuje alate poput OpenAI Playgrounda, koji vam omogućuje eksperimentiranje s uputama i fino podešavanje modela za specifične slučajeve korištenja. Nude opcije filtriranja sadržaja kako bi pomogli spriječiti generiranje neprimjerenog ili štetnog sadržaja.

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

Anthropic

Anthropic je još jedan važan igrač u području LLM-ova, čiji Claude modeli stječu popularnost zbog snažnih performansi i etičkih razmatranja. Anthropic se fokusira na razvoj sigurnih i odgovornih AI sustava, s jakim naglaskom na filtriranje sadržaja i izbjegavanje štetnih izlaznih podataka.

Anthropic-ov ekosustav uključuje Claude API, koji vam omogućuje integraciju modela u njihove aplikacije, kao i alate za inženjerstvo uputa i fino podešavanje. Također nude Claude Instant model, koji uključuje mogućnosti web pretraživanja za ažurnije i činjenično točnije odgovore.

Kada koristim Anthropic-ove modele izravno, 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 rasponu zadataka obrade prirodnog jezika. Google-ov ekosustav uključuje TensorFlow i Keras biblioteke, koje pružaju alate i okvire za izgradnju i treniranje modela strojnog učenja.

Google također nudi Cloud AI Platform, koji vam omogućuje jednostavno implementiranje i skaliranje njihovih modela u oblaku. Pružaju niz unaprijed istreniranih modela i API-ja za zadatke poput analize sentimenta, prepoznavanja entiteta i prevođenja.

Meta

Meta, prije poznata kao Facebook, duboko je uključena u razvoj velikih jezičnih modela, što je istaknuto njihovim izdavanjem modela poput LLaMA i OPT. Ovi modeli se ističu po svojim snažnim performansama u raznovrsnim jezičnim zadacima i uglavnom su dostupni putem kanala otvorenog koda, podržavajući Meta-inu predanost istraživanju i suradnji zajednice.

Meta-in ekosustav prvenstveno je izgrađen oko PyTorch-a, biblioteke otvorenog koda za strojno učenje koja je cijenjena zbog svojih dinamičkih računalnih mogućnosti i fleksibilnosti, olakšavajući inovativno AI istraživanje i razvoj.

Uz svoje tehničke ponude, Meta stavlja snažan naglasak na etični razvoj umjetne inteligencije. Implementiraju robusno filtriranje sadržaja i fokusiraju se na smanjenje pristranosti, u skladu sa svojim širim ciljevima sigurnosti i odgovornosti u primjenama umjetne inteligencije.

Cohere

Cohere je noviji sudionik u području LLM-ova, fokusirajući se na to da LLM-ovi budu pristupačniji i jednostavniji za korištenje od konkurencije. Njihov ekosustav uključuje

Cohere API, koji omogućuje pristup nizu unaprijed treniranih modela za zadatke poput generiranja teksta, klasifikacije i sažimanja.

Cohere također nudi alate za inženjering upita, fino podešavanje i filtriranje sadržaja. Naglašavaju privatnost i sigurnost podataka, s značajkama poput šifriranog spremanja podataka i kontrole pristupa.

Ollama

Ollama je samostalno hostana platforma koja korisnicima omogućuje upravljanje i implementaciju različitih velikih jezičnih modela (LLM-ova) lokalno na njihovim računalima, dajući im potpunu kontrolu nad njihovim AI modelima bez oslanjanja na vanjske cloud usluge. Ovo postavljanje je idealno za one koji prioritiziraju privatnost podataka i žele upravljati svojim AI operacijama interno.

Platforma podržava niz modela, uključujući verzije Llama, Phi, Gemma i Mistral, koji variraju u veličini i računalnim zahtjevima. Ollama olakšava preuzimanje i pokretanje ovih modela izravno iz naredbenog retka koristeći jednostavne naredbe poput `ollama run <model_name>`, a dizajnirana je za rad na različitim operativnim sustavima uključujući macOS, Linux i Windows.

Za developere koji žele integrirati modele otvorenog koda u svoje aplikacije bez korištenja udaljenog API-ja, Ollama nudi CLI za upravljanje životnim ciklusom modela slično alatima za upravljanje kontejnerima. Također podržava prilagođene konfiguracije i upite, omogućujući visok stupanj prilagodbe za prilagođavanje modela specifičnim potrebama ili slučajevima uporabe.

Ollama je posebno pogodna za tehnički obrazovane korisnike i developere zbog svog sučelja naredbenog retka i fleksibilnosti koju nudi u upravljanju i implementaciji AI modela. To je čini moćnim alatom za tvrtke i pojedince koji trebaju robusne AI mogućnosti bez kompromitiranja sigurnosti i kontrole.

Multi-Model Platforme

Dodatno, postoje pružatelji usluga koji hostaju širok spektar modela otvorenog koda, kao što su Together.ai i Groq. Ove platforme nude fleksibilnost i prilagodbu, omogućujući vam pokretanje i, u nekim slučajevima, čak i fino podešavanje modela otvorenog koda prema vašim specifičnim potrebama. Na primjer, Together.ai pruža pristup nizu LLM-ova otvorenog koda, omogućujući korisnicima eksperimentiranje s različitim modelima i konfiguracijama. Groq se fokusira na pružanje ultra visoko-performansnog dovršavanja koje se u vrijeme pisanja ove knjige čini gotovo magičnim

Odabir LLM pružatelja usluga

Pri odabiru LLM pružatelja usluga, trebali biste razmotriti faktore poput:

- **Cijene**: Različiti pružatelji nude različite modele cijena, od plaćanja po korištenju do pretplatničkih planova. Važno je uzeti u obzir očekivanu uporabu i proračun pri odabiru pružatelja.
- **Performanse**: Performanse LLM-ova mogu značajno varirati između pružatelja, pa je važno testirati modele na specifičnim slučajevima uporabe prije donošenja odluke.
- **Filtriranje sadržaja**: Ovisno o aplikaciji, filtriranje sadržaja može biti kritično razmatranje. Neki pružatelji nude robusnije opcije filtriranja sadržaja od drugih.
- **Privatnost podataka**: Ako aplikacija rukuje osjetljivim korisničkim podacima, važno je odabrati pružatelja s jakim praksama privatnosti i sigurnosti podataka.
- **Prilagodba**: Neki pružatelji nude više fleksibilnosti u smislu finog podešavanja i prilagođavanja modela za specifične slučajeve uporabe.

U konačnici, izbor LLM pružatelja ovisi o specifičnim zahtjevima i ograničenjima aplikacije. Pažljivim vrednovanjem opcija i razmatranjem faktora poput cijene, performansi i privatnosti podataka, možete odabrati pružatelja koji najbolje odgovara vašim potrebama.

Također je vrijedno napomenuti da se LLM područje konstantno razvija, s novim pružateljima i modelima koji se redovito pojavljuju. Trebali biste biti u toku s najnovijim razvojima i biti otvoreni za istraživanje novih opcija kako postaju dostupne.

OpenRouter

Kroz ovu knjigu oslanjat ću se isključivo na OpenRouter kao moj odabrani API pružatelj. Razlog je jednostavan: to je jedinstveno mjesto za sve najpopularnije komercijalne modele i modele otvorenog koda. Ako jedva čekate započeti s AI programiranjem, jedno od najboljih mjesta za početak je moja vlastita OpenRouter Ruby biblioteka.

Razmišljanje o performansama

Pri ugradnji jezičnih modela u aplikacije, performanse su ključno razmatranje. Performanse jezičnog modela mogu se mjeriti u smislu njegove *latencije* (vrijeme potrebno za generiranje odgovora) i *propusnosti* (broj zahtjeva koje može obraditi po jedinici vremena).

Vrijeme do prvog tokena (TTFT) je još jedna važna metrika performansi, posebno relevantna za chatbotove i aplikacije koje zahtijevaju interaktivne odgovore u stvarnom vremenu. TTFT mjeri latenciju od trenutka kada je primljen korisnički zahtjev do trenutka kada je generirana prva riječ (ili token) odgovora. Ova metrika je ključna za održavanje neometanog i privlačnog korisničkog iskustva, jer zakašnjeli odgovori mogu dovesti do frustracije i odustajanja korisnika.

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

Nekoliko faktora može utjecati na performanse jezičnog modela, uključujući:

Broj parametara: Veći modeli s više parametara općenito zahtijevaju više računalnih resursa i mogu imati veću latenciju i manju propusnost u usporedbi s manjim modelima.

Hardver: Performanse jezičnog modela mogu značajno varirati ovisno o hardveru na kojem se izvodi. Pružatelji usluga u oblaku nude GPU i TPU instance optimizirane za strojno učenje, koje mogu značajno ubrzati izvođenje modela.

Jedna od lijepih stvari kod OpenRoutera je što za mnoge modele koje nudi dobivate izbor pružatelja usluga u oblaku s različitim profilima performansi i troškova.

Kvantizacija: Tehnike kvantizacije mogu se koristiti za smanjenje memorijskog otiska i računalnih zahtjeva modela predstavljanjem težina i aktivacija tipovima podataka niže preciznosti. Ovo može poboljšati performanse bez značajnog žrtvovanja kvalitete. Kao razvojni programer aplikacija, vjerojatno se nećete baviti treniranjem vlastitih modela na različitim razinama kvantizacije, ali dobro je barem biti upoznat s terminologijom.

Grupiranje: Obrada više zahtjeva istovremeno u grupama može poboljšati propusnost amortiziranjem režijskih troškova učitavanja modela i prijenosa podataka.

Predmemoriranje: Predmemoriranje rezultata često korištenih upita ili ulaznih sekvenci može smanjiti broj zahtjeva za zaključivanjem i poboljšati ukupne performanse.

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

Također je vrijedno razmotriti kompromise između performansi modela i drugih faktora poput troškova, fleksibilnosti i lakoće integracije. Na primjer, korištenje manjeg, jeftinijeg modela s nižom latencijom može biti poželjnije za aplikacije koje zahtijevaju odgovore u stvarnom vremenu, dok veći, snažniji model može biti prikladniji za grupnu obradu ili složene zadatke rasuđivanja.

Eksperimentiranje s različitim LLM modelima

Odabir LLM-a rijetko je trajna odluka. Kako se redovito objavljuju novi i poboljšani modeli, dobro je graditi aplikacije na modularan način koji omogućuje zamjenu različitih jezičnih modela tijekom vremena. Upiti i skupovi podataka često se mogu ponovno koristiti kroz različite modele uz minimalne promjene. To vam omogućuje da iskoristite najnovija dostignuća u modeliranju jezika bez potrebe za potpunim redizajnom aplikacija.

Mogućnost jednostavnog prebacivanja između širokog raspona izbora modela još je jedan razlog zašto volim OpenRouter.

Pri nadogradnji na novi jezični model, važno je temeljito testirati i validirati njegove performanse i kvalitetu izlaza kako bi se osiguralo da zadovoljava zahtjeve aplikacije. To može uključivati ponovno treniranje ili fino podešavanje modela na domenski specifičnim podacima, kao i ažuriranje svih nizvodnih komponenti koje ovise o izlazima modela.

Dizajniranjem aplikacija s fokusom na performanse i modularnost, možete stvoriti skalabilne, učinkovite i budućnosti prilagođene sustave koji se mogu prilagoditi brzo razvijajućem krajoliku tehnologije jezičnog modeliranja.

Složeni AI sustavi

Prije završetka našeg uvoda, vrijedi spomenuti da su prije 2023. godine i eksplozije interesa za generativni AI potaknutog ChatGPT-om, tradicionalni AI pristupi obično ovisili o integraciji pojedinačnih, zatvorenih modela. Nasuprot tome, *Složeni AI sustavi* koriste složene cjevovode međusobno povezanih komponenti koje rade zajedno kako bi postigle inteligentno ponašanje.

U svojoj srži, složeni AI sustavi sastoje se od više modula, od kojih je svaki dizajniran za izvođenje specifičnih zadataka ili funkcija. Ovi moduli mogu uključivati generatore, dohvatnike, rangirnike, klasifikatore i razne druge specijalizirane komponente. Razbijanjem cjelokupnog sustava na manje, fokusirane jedinice, razvojni programeri mogu stvoriti fleksibilnije, skalabilnije i održivije AI arhitekture.

Jedna od ključnih prednosti složenih AI sustava je njihova sposobnost kombiniranja snaga različitih AI tehnika i modela. Na primjer, sustav može koristiti veliki jezični model (LLM) za razumijevanje i generiranje prirodnog jezika, dok istovremeno koristi zaseban model za dohvaćanje informacija ili donošenje odluka temeljeno na pravilima. Ovaj modularni pristup omogućuje vam odabir najboljih alata i tehnika za svaki specifični zadatak, umjesto oslanjanja na jedinstveno rješenje za sve situacije.

Međutim, izgradnja složenih AI sustava također predstavlja jedinstvene izazove. Posebno, osiguravanje opće koherentnosti i dosljednosti ponašanja sustava zahtijeva robusne mehanizme testiranja, praćenja i upravljanja.

Pojava moćnih LLM-ova poput GPT-4 omogućuje nam lakše eksperimentiranje sa složenim AI sustavima nego ikad prije, jer su ovi napredni modeli sposobni obavljati višestruke uloge unutar složenog sustava, kao što su klasifikacija, rangiranje i generiranje, uz svoje sposobnosti razumijevanja prirodnog jezika. Ova svestranost omogućuje programerima brzo prototipiranje i iteriranje na arhitekturama složenih AI sustava, otvarajući nove mogućnosti za razvoj inteligentnih aplikacija.

Obrasci implementacije za složene AI sustave

Složeni AI sustavi mogu se implementirati koristeći različite obrasce, od kojih je svaki dizajniran za rješavanje specifičnih zahtjeva i slučajeva uporabe. Istražimo četiri uobičajena obrasca implementacije: Pitanja i odgovori, Višeagentni/Agentski rješavatelji problema, Konverzacijski AI i CoPiloti.

Pitanja i odgovori

Sustavi pitanja i odgovora (Q&A) fokusiraju se na pružanje dohvaćanja informacija koje je poboljšano sposobnostima razumijevanja AI modela kako bi funkcionirali kao više od običnog pretraživača. Kombiniranjem moćnih jezičnih modela s vanjskim izvorima znanja koristeći Dohvaćanjem potpomognutu generaciju (RAG), sustavi pitanja i odgovora izbjegavaju halucinacije i pružaju točne i kontekstualno relevantne odgovore na upite korisnika.

Ključne komponente Q&A sustava temeljenog na LLM-u uključuju:

- **Razumijevanje i preformuliranje upita**: Analiza korisničkih upita i njihovo preformuliranje kako bi bolje odgovarali osnovnim izvorima znanja.
- **Dohvaćanje znanja**: Dohvaćanje relevantnih informacija iz strukturiranih ili nestrukturiranih izvora podataka na temelju preformuliranog upita.
- **Generiranje odgovora**: Generiranje koherentnih i informativnih odgovora integrirajući dohvaćeno znanje s generativnim sposobnostima jezičnog modela.

RAG podsustavi posebno su važni u Q&A domenama gdje je pružanje točnih i ažurnih informacija ključno, kao što su korisnička podrška, upravljanje znanjem ili obrazovne aplikacije.

Višeagentni/Agentski rješavatelji problema

Višeagentni, također poznati kao *Agentski* sustavi, sastoje se od više autonomnih agenata koji zajedno rade na rješavanju složenih problema. Svaki agent ima specifičnu ulogu, skup vještina i pristup relevantnim alatima ili izvorima informacija. Suradnjom i razmjenom informacija, ovi agenti mogu rješavati zadatke koji bi bili teški ili nemogući za jednog agenta.

Ključni principi višeagentnih rješavatelja problema uključuju:

- **Specijalizaciju**: Svaki agent fokusira se na specifični aspekt problema, koristeći svoje jedinstvene sposobnosti i znanje.
- **Suradnju**: Agenti komuniciraju i koordiniraju svoje akcije kako bi postigli zajednički cilj, često putem razmjene poruka ili dijeljene memorije.
- **Prilagodljivost**: Sustav se može prilagoditi promjenjivim uvjetima ili zahtjevima prilagođavanjem uloga i ponašanja pojedinačnih agenata.

Višeagentni sustavi posebno su pogodni za aplikacije koje zahtijevaju distribuirano rješavanje problema, kao što su optimizacija opskrbnog lanca, upravljanje prometom ili planiranje odgovora na hitne situacije.

Konverzacijski AI

Konverzacijski AI sustavi omogućuju interakcije prirodnim jezikom između korisnika i inteligentnih agenata. Ovi sustavi kombiniraju sposobnosti razumijevanja prirodnog jezika, upravljanja dijalogom i generiranja jezika kako bi pružili angažirajuća i personalizirana konverzacijska iskustva.

Glavne komponente konverzacijskog AI sustava uključuju:

- **Prepoznavanje namjere**: Identificiranje korisnikove namjere na temelju njihovog unosa, kao što su postavljanje pitanja, upućivanje zahtjeva ili izražavanje osjećaja.
- **Izdvajanje entiteta**: Izdvajanje relevantnih entiteta ili parametara iz korisničkog unosa, kao što su datumi, lokacije ili nazivi proizvoda.
- **Upravljanje dijalogom**: Održavanje stanja razgovora, određivanje primjerenog odgovora na temelju korisnikove namjere i konteksta te upravljanje višestrukim interakcijama.
- **Generiranje odgovora**: Generiranje odgovora nalik ljudskima koristeći jezične modele, predloške ili metode temeljene na dohvaćanju.

Konverzacijski AI sustavi često se koriste u chatbotovima za korisničku podršku, virtualnim asistentima i sučeljima upravljanim glasom. Kao što je ranije spomenuto,

većina pristupa, obrazaca i primjera koda u ovoj knjizi izravno su izvučeni iz mog rada na velikom konverzacijskom AI sustavu zvanom Olympia.

Kopiloti

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

Ključne značajke Kopilota uključuju:

- **Personalizaciju**: Prilagođavanje individualnim korisničkim preferencijama, radnim tijekovima i stilovima komunikacije.
- **Proaktivnu pomoć**: Predviđanje korisničkih potreba i nuđenje relevantnih prijedloga ili akcija bez izričitih upita.
- **Kontinuirano učenje**: Poboljšavanje performansi tijekom vremena učenjem iz korisničkih povratnih informacija, interakcija i podataka.

Kopiloti se sve više koriste u raznim domenama, kao što su razvoj softvera (npr. dovršavanje koda i otkrivanje grešaka), kreativno pisanje (npr. prijedlozi sadržaja i uređivanje), i analiza podataka (npr. uvidi i preporuke za vizualizaciju)

Ovi obrasci implementacije pokazuju svestranost i potencijal složenih AI sustava. Razumijevanjem karakteristika i slučajeva uporabe svakog obrasca, možete donositi informirane odluke pri dizajniranju i implementaciji inteligentnih aplikacija. Iako ova knjiga nije specifično o implementaciji složenih AI sustava, mnogi, ako ne i svi isti pristupi i obrasci primjenjuju se na integraciju zasebnih AI komponenti unutar inače tradicionalnog razvoja aplikacija.

Uloge u složenim AI sustavima

Složeni AI sustavi izgrađeni su na temelju međusobno povezanih modula, od kojih je svaki dizajniran za izvršavanje specifične uloge. Ovi moduli rade zajedno kako bi stvorili inteligentna ponašanja i riješili složene probleme. Korisno je biti upoznat s ovim ulogama kada razmišljate o tome gdje biste mogli implementirati ili zamijeniti dijelove svoje aplikacije zasebnim AI komponentama.

Generator

Generatori su odgovorni za proizvodnju novih podataka ili sadržaja na temelju naučenih obrazaca ili ulaznih upita. AI svijet ima mnogo različitih vrsta generatora, ali u kontekstu jezičnih modela koji su prikazani u ovoj knjizi, generatori mogu stvarati tekst nalik ljudskom, dovršavati djelomične rečenice ili generirati odgovore na korisničke upite. Oni igraju ključnu ulogu u zadacima kao što su stvaranje sadržaja, generiranje dijaloga i proširivanje podataka.

Dohvatitelj

Dohvatitelji se koriste za pretraživanje i izvlačenje relevantnih informacija iz velikih skupova podataka ili baza znanja. Oni koriste tehnike poput semantičkog pretraživanja, podudaranja ključnih riječi ili vektorske sličnosti kako bi pronašli najrelevantnije podatke na temelju zadanog upita ili konteksta. Dohvatitelji su ključni za zadatke koji zahtijevaju brz pristup specifičnim informacijama, kao što su odgovaranje na pitanja, provjera činjenica ili preporuka sadržaja.

Rangiratelj

Rangiratelji su odgovorni za uređivanje ili prioritiziranje skupa stavki na temelju određenih kriterija ili ocjena relevantnosti. Oni dodjeljuju težine ili ocjene svakoj stavci i zatim ih sortiraju u skladu s tim. Rangiratelji se često koriste u tražilicama, sustavima za

preporuke ili bilo kojoj aplikaciji gdje je ključno predstavljanje najrelevantnijih rezultata korisnicima.

Klasifikator

Klasifikatori se koriste za kategorizaciju ili označavanje podatkovnih točaka na temelju unaprijed definiranih klasa ili kategorija. Oni uče iz označenih podataka za treniranje i zatim predviđaju klasu novih, neviđenih primjera. Klasifikatori su temeljni za zadatke poput analize sentimenta, otkrivanja neželjene pošte ili prepoznavanja slika, gdje je cilj dodijeliti specifičnu kategoriju svakom ulazu.

Alati i Agenti

Uz ove osnovne uloge, složeni AI sustavi često uključuju alate i agente za poboljšanje njihove funkcionalnosti i prilagodljivosti:

- **Alati**: Alati su zasebne softverske komponente ili API-ji koji izvršavaju specifične akcije ili izračune. Mogu ih pozivati drugi moduli, poput generatora ili dohvatitelja, za izvršavanje podzadataka ili prikupljanje dodatnih informacija. Primjeri alata uključuju web tražilice, kalkulatore ili biblioteke za vizualizaciju podataka.
- **Agenti**: Agenti su autonomni entiteti koji mogu percipirati svoje okruženje, donositi odluke i poduzimati akcije za postizanje specifičnih ciljeva. Često se oslanjaju na kombinaciju različitih AI tehnika, kao što su planiranje, zaključivanje i učenje, kako bi učinkovito djelovali u dinamičnim ili neizvjesnim uvjetima. Agenti se mogu koristiti za modeliranje složenih ponašanja ili za koordinaciju akcija više modula unutar složenog AI sustava.

U čistom složenom AI sustavu, interakcija između ovih komponenti orkestrirana je kroz dobro definirane sučelja i komunikacijske protokole. Podaci teku između modula, gdje izlaz jedne komponente služi kao ulaz za drugu. Ova modularna arhitektura omogućuje

fleksibilnost, skalabilnost i održivost, jer se pojedinačne komponente mogu ažurirati, zamijeniti ili proširiti bez utjecaja na cijeli sustav.

Korištenjem snage ovih komponenti i njihovih interakcija, složeni AI sustavi mogu se uhvatiti u koštac sa složenim, stvarnim problemima koji zahtijevaju 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 korišteni u složenim AI sustavima mogu primijeniti za stvaranje inteligentnih, prilagodljivih i korisnički orijentiranih aplikacija.

U sljedećim poglavljima Dijela 1, dublje ćemo zaroniti u temeljne pristupe i tehnike za integraciju AI komponenti u vaš proces razvoja aplikacija. Od inženjeringa upita i generiranja potpomognutog dohvaćanjem do samoobnavljajućih podataka i inteligentne orkestracije radnih tijekova, pokrit ćemo širok raspon obrazaca i najboljih praksi kako bismo vam pomogli izgraditi najsuvremenije aplikacije pokretane AI-jem.

Prvi dio: Temeljni pristupi i tehnike

Ovaj dio knjige predstavlja različite načine integracije umjetne inteligencije u vaše aplikacije. Poglavlja obuhvaćaju niz povezanih pristupa i tehnika, od konceptualnijih ideja poput Sužavanja Puta i Retrieval Augmented Generation, pa sve do ideja za programiranje vlastitog apstrakcijskog sloja povrh LLM API-ja za dovršavanje razgovora.

Cilj ovog dijela knjige je pomoći vam razumjeti vrste ponašanja koje možete implementirati s umjetnom inteligencijom, prije nego što se dublje upustite u specifične implementacijske obrasce koji su fokus Drugog dijela.

Pristupi u Prvom dijelu temelje se na idejama koje sam koristio u svojem kodu, klasičnim obrascima arhitekture poslovnih aplikacija i integracije, kao i metaforama koje sam koristio pri objašnjavanju mogućnosti umjetne inteligencije drugim ljudima, uključujući i poslovne dionike koji nisu tehnički stručnjaci.

Suziti Put

“Suziti put” odnosi se na usmjeravanje UI-ja na zadatak koji je pred njim. Koristim to kao mantru kad god me frustrira UI koji se ponaša “glupo” ili na neočekivane načine. Mantra me podsjeća da je neuspjeh vjerojatno moja krivica i da bih vjerojatno trebao još više suziti put.

Potreba za sužavanjem puta proizlazi iz ogromne količine znanja sadržanog u velikim jezičnim modelima, posebice u svjetski poznatim modelima kao što su oni od OpenAI-ja i Anthropica koji doslovno imaju bilijune parametara.

Pristup tako širokom rasponu znanja nedvojbeno je moćan i proizvodi emergentno ponašanje poput teorije uma i sposobnosti rasuđivanja na način sličan ljudskom. Međutim, ta zapanjujuća količina informacija također predstavlja izazove kada je riječ o generiranju preciznih i točnih odgovora na specifične upite, posebno ako ti upiti trebaju pokazivati deterministično ponašanje koje se može integrirati s "normalnim" razvojem softvera i algoritmima.

Nekoliko čimbenika dovodi do tih izazova.

Preopterećenost informacijama: Veliki jezični modeli trenirani su na masivnim količinama podataka koji obuhvaćaju različite domene, izvore i vremenska razdoblja. Ovo opsežno znanje omogućuje im da se bave raznolikim temama i generiraju odgovore temeljene na širokom razumijevanju svijeta. Međutim, kada se suoči s određenim upitom, model se može mučiti s filtriranjem irelevantnih, kontradiktornih ili zastarjelih/obsoletnih informacija, što dovodi do odgovora koji nemaju fokus ili točnost. Ovisno o tome što pokušavate postići, sama količina *kontradiktornih* informacija dostupnih modelu lako može nadjačati njegovu sposobnost da pruži odgovor ili ponašanje koje tražite.

Kontekstualna dvosmislenost: S obzirom na ogromni *latentni prostor* znanja, veliki jezični modeli mogu naići na dvosmislenost pri pokušaju razumijevanja *konteksta* vašeg upita. Bez pravilnog sužavanja ili usmjeravanja, model može generirati odgovore koji su tangencijalno povezani, ali nisu izravno relevantni za vaše namjere. Ova vrsta neuspjeha dovodi do odgovora koji su izvan teme, nedosljedni ili ne zadovoljavaju vaše navedene potrebe. U ovom slučaju, sužavanje puta odnosi se na *razjašnjavanje* konteksta, osiguravajući da kontekst koji pružate uzrokuje da se model fokusira samo na najrelevantnije informacije u svojem osnovnom znanju.

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

Vremenske nedosljednosti: Budući da su jezični modeli trenirani na podacima koji su nastali u različitim vremenskim razdobljima, mogu posjedovati znanje koje je zastarjelo, prevladano ili više nije točno. Na primjer, informacije o trenutnim događajima, znanstvenim otkrićima ili tehnološkim naprecima možda su se razvile od trenutka prikupljanja podataka za treniranje modela. Bez sužavanja puta kako bi se dao prioritet novijim i pouzdanijim izvorima, model može generirati odgovore temeljene na zastarjelim ili netočnim informacijama, što dovodi do netočnosti i nedosljednosti u njegovim izlazima.

Domenski specifične nijanse: Različite domene i područja imaju svoju specifičnu terminologiju, konvencije i baze znanja. Razmislite o praktički bilo kojem TLS-u (Tro-Slovnoj Skraćenici) i shvatit ćete da većina njih ima više od jednog značenja. Na primjer, MSK može se odnositi na Amazon-ov Managed Streaming for Apache Kafka, Memorial Sloan Kettering Cancer Center ili ljudski MuskuloSKeletni sustav.

Kada upit zahtijeva stručnost u određenoj domeni, generičko znanje velikog jezičnog modela možda neće biti dovoljno za pružanje točnih i nijansiranih odgovora. Sužavanje puta fokusiranjem na domenski specifične informacije, bilo kroz inženjerstvo upita ili generiranje potpomognuto dohvaćanjem, omogućuje modelu da generira odgovore koji su više usklađeni sa zahtjevima i očekivanjima vaše specifične domene.

Latentni prostor: Neshvatljivo prostran

Kada spominjem “latentni prostor” jezičnog modela, govorim o ogromnom, višedimenzionalnom krajoliku znanja i informacija koje je model naučio tijekom procesa treniranja. To je poput skrivenog carstva unutar neuronskih mreža modela, gdje su pohranjeni svi uzorci, asocijacije i reprezentacije jezika.

Zamislite da istražujete prostrano, neistraženo područje ispunjeno bezbrojnim međusobno povezanim čvorovima. Svaki čvor predstavlja djelić informacije, koncept ili odnos koji je model naučio. Dok se krećete kroz taj prostor, otkrit ćete da su neki

čvorovi bliže jedan drugome, što ukazuje na snažnu povezanost ili sličnost, dok su drugi udaljeniji, što sugerira slabiju ili udaljeniju vezu.

Izazov s latentnim prostorom je taj što je nevjerojatno složen i visokodimenzionalan. Zamislite ga velikim poput našeg fizičkog svemira, s njegovim nakupinama galaksija i ogromnim, nezamislivim udaljenostima praznog prostora između njih.

Budući da sadrži tisuće dimenzija, latentni prostor nije izravno vidljiv niti ga ljudi mogu interpretirati. To je apstraktna reprezentacija koju model interno koristi za obradu i generiranje jezika. Kada modelu date ulazni prompt, on ga u biti mapira na određenu lokaciju unutar latentnog prostora. Model zatim koristi okolne informacije i veze u tom prostoru za generiranje odgovora.

Stvar je u tome što je model naučio ogromnu količinu informacija iz svojih podataka za treniranje, i nisu sve relevantne ili točne za određeni zadatak. Zato sužavanje puta postaje tako važno. Pružanjem jasnih uputa, primjera i konteksta u svojim promptovima, u biti vodite model da se usredotoči na specifična područja unutar latentnog prostora koja su najrelevantnija za željeni rezultat.

Drugi način razmišljanja o tome je kao korištenje reflektora u potpuno mračnom muzeju. Ako ste ikada posjetili Louvre ili Metropolitan Museum of Art, onda je to razmjer o kojem govorim. Latentni prostor je muzej, ispunjen nebrojenim predmetima i detaljima. Vaš prompt je reflektor koji osvjetljava određena područja i usmjerava pozornost modela na najvažnije informacije. Bez tog vodstva, model može besciljno lutati kroz latentni prostor, skupljajući usput irelevantne ili kontradiktorne informacije.

Dok radite s jezičnim modelima i oblikujete svoje promptove, imajte na umu koncept latentnog prostora. Vaš cilj je učinkovito navigirati kroz taj ogromni krajolik znanja, usmjeravajući model prema najrelevantnijim i najtočnijim informacijama za vaš zadatak. Sužavanjem puta i pružanjem jasnog vodstva možete otključati puni potencijal latentnog prostora modela i generirati kvalitetne, koherentne odgovore.

Iako prethodni opisi jezičnih modela i latentnog prostora kroz koji navigiraju mogu djelovati pomalo magično ili apstraktno, važno je razumjeti da promptovi nisu čarolije

ili čarobne formule. Način na koji jezični modeli rade temelji se na principima linearne algebre i teorije vjerojatnosti.

U svojoj srži, jezični modeli su probabilistički modeli teksta, slično kao što je Gaussova krivulja statistički model podataka. Treniraju se kroz proces koji se zove autoregresivno modeliranje, gdje model uči predvidjeti vjerojatnost sljedeće riječi u nizu na temelju riječi koje joj prethode. Tijekom treninga, model počinje sa slučajnim težinama i postupno ih prilagođava kako bi dodijelio veće vjerojatnosti tekstu koji nalikuje stvarnim uzorcima na kojima je treniran.

Međutim, razmišljanje o jezičnim modelima kao jednostavnim statističkim modelima, poput linearne regresije, ne pruža najbolju intuiciju za razumijevanje njihovog ponašanja. Prikladnija analogija je razmišljati o njima kao o probabilističkim programima, koji su modeli koji omogućuju manipulaciju slučajnim varijablama i mogu predstavljati složene statističke odnose.

Probabilistički programi mogu se predstaviti grafičkim modelima, koji pružaju vizualni način razumijevanja ovisnosti i odnosa između varijabli u modelu. Ova perspektiva može ponuditi vrijedne uvide u rad složenih modela za generiranje teksta poput GPT-4 i Claudea.

U radu "Language Model Cascades" autora Dohana i suradnika, autori ulaze u detalje o tome kako se probabilistički programi mogu primijeniti na jezične modele. Pokazuju kako se ovaj okvir može koristiti za razumijevanje ponašanja ovih modela i usmjeravanje razvoja učinkovitijih strategija promptanja.

Jedan ključni uvid iz ove probabilističke perspektive jest da jezični model u biti stvara portal u alternativni svemir gdje željeni dokumenti postoje. Model dodjeljuje težine svim mogućim dokumentima na temelju njihove vjerojatnosti, učinkovito sužavajući prostor mogućnosti kako bi se usredotočio na najrelevantnije.

To nas vraća na središnju temu "sužavanja puta". Primarni cilj promptanja je uvjetovati probabilistički model na način koji fokusira masu njegovih predviđanja, usredotočujući se na specifične informacije ili ponašanje koje želimo izazvati. Pružanjem pažljivo

oblikovanih promptova možemo voditi model da učinkovitije navigira latentnim prostorom i generira rezultate koji su relevantniji i koherentniji.

Međutim, važno je imati na umu da je jezični model u konačnici ograničen informacijama na kojima je treniran. Iako može generirati tekst koji je sličan postojećim dokumentima ili kombinirati ideje na nove načine, ne može stvoriti potpuno nove informacije ni iz čega. Na primjer, ne možemo očekivati da model pruži lijek za rak ako takav lijek nije otkriven i dokumentiran u njegovim podacima za treniranje.

Umjesto toga, snaga modela leži u njegovoj sposobnosti pronalaženja i sintetiziranja informacija koje su slične onome što mu zadajemo kao upit. Razumijevanjem probabilističke prirode ovih modela i načina na koji se upiti mogu koristiti za uvjetovanje njihovih izlaza, možemo učinkovitije iskoristiti njihove mogućnosti za generiranje vrijednih uvida i sadržaja.

Pogledajmo upite u nastavku. U prvom, sama riječ "Merkur" mogla bi se odnositi na planet, kemijski element ili rimskog boga, ali najvjerojatniji je planet. Zaista, GPT-4 daje opširan odgovor koji počinje s *Merkur je najmanji i najbliži planet Sunčevom sustavu...*. Drugi upit posebno se odnosi na kemijski element. Treći se odnosi na lik iz rimske mitologije, poznat po svojoj 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 riječi, u potpunosti smo promijenili način na koji AI reagira. Kao što ćete kasnije naučiti u knjizi, napredni trikovi u inženjerstvu uputa poput n-shot promptinga, strukturiranog unosa/izlaza i lanca razmišljanja su samo domišljati načini uvjetovanja izlaza modela.

Dakle, u konačnici, umjetnost inženjerstva uputa svodi se na razumijevanje kako navigirati kroz prostrano probabilističko područje znanja jezičnog modela kako bismo suzili put do specifične informacije ili ponašanja koje tražimo.

Za čitatelje s čvrstim razumijevanjem napredne matematike, temeljenje vašeg razumijevanja ovih modela na principima teorije vjerojatnosti i linearne algebre definitivno može pomoći! Za ostale koji žele razviti učinkovite strategije za dobivanje željenih rezultata, držat ćemo se intuitivnijih pristupa.

Kako se put "sužava"

Kako bismo se suočili s izazovima prevelikog znanja, koristimo tehnike koje pomažu usmjeriti proces generiranja jezičnog modela i fokusirati njegovu pažnju na najrelevantnije i najtočnije informacije.

Evo najznačajnijih tehnika, u preporučenom redoslijedu, odnosno, trebali biste prvo isprobati inženjerstvo uputa, zatim RAG, i konačno, ako morate, fino podešavanje.

Inženjerstvo uputa Najosnovniji pristup je izrada uputa koje uključuju specifične instrukcije, ograničenja ili primjere za usmjeravanje generiranja odgovora modela. Ovo poglavlje pokriva osnove inženjerstva uputa u sljedećem odjeljku, a mnoge specifične obrasce inženjerstva uputa obrađujemo u drugom dijelu knjige. Ti obrasci uključuju Destilaciju uputa, tehniku koja se fokusira na rafiniranje i optimiziranje uputa kako bi se izvukle informacije koje AI smatra najrelevantnijima i najsažetijima.

Kontekstualno proširenje. Dinamičko dohvaćanje relevantnih informacija iz vanjskih baza znanja ili dokumenata kako bi se modelu osigurao fokusirani kontekst u trenutku kada mu se daje uputa. Popularne tehnike kontekstualnog proširenja uključuju Dohvatom potpomognuto generiranje (RAG) Takozvani "online modeli" poput onih koje pruža Perplexity mogu proširiti svoj kontekst rezultatima pretraživanja interneta u stvarnom vremenu.

Unatoč njihovoj snazi, LLM-ovi nisu trenirani na vašim jedinstvenim skupovima podataka, koji mogu biti privatni ili specifični za problem koji pokušavate riješiti. Tehnike kontekstualnog proširenja omogućuju vam da LLM-ovima date pristup podacima iza API-ja, u SQL bazama podataka ili zarobljenim u PDF-ovima i prezentacijama.

Fino podešavanje ili prilagodba domeni Treniranje modela na skupovima podataka specifičnim za domenu kako bi se specijaliziralo njegovo znanje i sposobnosti generiranja za određeni zadatak ili područje.

Smanjivanje temperature

Temperatura je *hiperparametar* koji se koristi u transformer-baziranim jezičnim modelima za kontrolu nasumičnosti i kreativnosti generiranog teksta. To je vrijednost između 0 i 1, gdje niže vrijednosti čine izlaz fokusiranijim i determinističkijim, dok više vrijednosti čine izlaz raznovrsnijim i nepredvidljivijim.

Kada je temperatura postavljena na 1, jezični model generira tekst na temelju pune distribucije vjerojatnosti sljedećeg tokena, omogućujući kreativnije i raznovrsnije odgovore. Međutim, to također može dovesti do toga da model generira tekst koji je manje relevantan ili koherentan.

S druge strane, kada je temperatura postavljena na 0, jezični model uvijek odabire token s najvišom vjerojatnošću, efektivno "sužavajući svoj put". Gotovo 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 *slijedi upute*, obrati pažnju na funkcije koje su mu pružene ili jednostavno trebate točnije i relevantnije odgovore od onih koje dobivate.

Na primjer, ako gradite chatbota koji treba pružati činjenične informacije, možda ćete željeti postaviti temperaturu na nižu vrijednost kako biste osigurali da su odgovori precizniji i više vezani uz temu. Suprotno tome, ako gradite asistenta za kreativno

pisanje, možda ćete željeti postaviti temperaturu na višu vrijednost kako biste potaknuli raznovrsnije i maštovitije rezultate.

Hiperparametri: Gumbi i prekidači zaključivanja

Kada radite s jezičnim modelima, često ćete naići na pojam "hiperparametri". U kontekstu zaključivanja (tj. kada koristite model za generiranje odgovora), hiperparametri su poput gumba i prekidača koje možete podešavati kako biste kontrolirali ponašanje i izlaz modela.

Zamislite to kao podešavanje postavki na složenom stroju. Baš kao što biste mogli okrenuti gumb za kontrolu temperature ili prebaciti prekidač za promjenu načina rada, hiperparametri vam omogućuju fino podešavanje načina na koji jezični model obrađuje i generira tekst.

Neki uobičajeni hiperparametri s kojima ćete se susresti tijekom zaključivanja uključuju:

- **Temperatura**: Kao što je upravo spomenuto, ovaj parametar kontrolira nasumičnost i kreativnost generiranog teksta. Viša temperatura dovodi do raznolikijih i nepredvidljivijih izlaza, dok niža temperatura rezultira fokusiranijim i determinističkijim odgovorima.

- **Top-p (jezgreno) uzorkovanje**: Ovaj parametar kontrolira odabir najmanjeg skupa tokena čija kumulativna vjerojatnost prelazi određeni prag (p). Omogućuje raznolikije izlaze uz održavanje koherentnosti.

- **Top-k uzorkovanje**: Ova tehnika odabire *k* najvjerojatnijih sljedećih tokena i preraspodjeljuje masu vjerojatnosti među njima. Može pomoći u sprječavanju modela da generira tokene niske vjerojatnosti ili irelevantne tokene.

- **Kazne učestalosti** i **prisutnosti**: Ovi parametri kažnjavaju model za prečesto ponavljanje istih riječi ili fraza (kazna učestalosti) ili za generiranje riječi koje

nisu prisutne u ulaznom upitu (kazna prisutnosti). Podešavanjem ovih vrijednosti možete potaknuti model da proizvodi raznolikije i relevantnije izlaze.

- **Maksimalna duljina**: Ovaj hiperparametar postavlja gornju granicu broja tokena (riječi ili podriječi) koje model može generirati u jednom odgovoru. Pomaže u kontroli opširnosti i sažetosti generiranog teksta.

Dok eksperimentirate s različitim postavkama hiperparametara, otkrit ćete da čak i male prilagodbe mogu značajno utjecati na izlaz modela. To je poput finog podešavanja recepta – prstohvat više soli ili malo duže vrijeme kuhanja mogu činiti svu razliku u konačnom jelu.

Ključno je razumjeti kako svaki hiperparametar utječe na ponašanje modela i pronaći pravu ravnotežu za vaš specifični zadatak. Ne bojte se eksperimentirati s različitim postavkama i vidjeti kako utječu na generirani tekst. S vremenom ćete razviti intuiciju o tome koje hiperparametre treba prilagoditi i kako postići željene rezultate.

Kombiniranjem korištenja ovih parametara s inženjerstvom uputa, generiranjem potpomognutim dohvaćanjem i finim podešavanjem, možete učinkovito suziti put i voditi jezični model da generira točnije, relevantnije i vrijednije odgovore za svoj specifični slučaj uporabe.

Sirovi modeli nasuprot modelima podešenim za upute

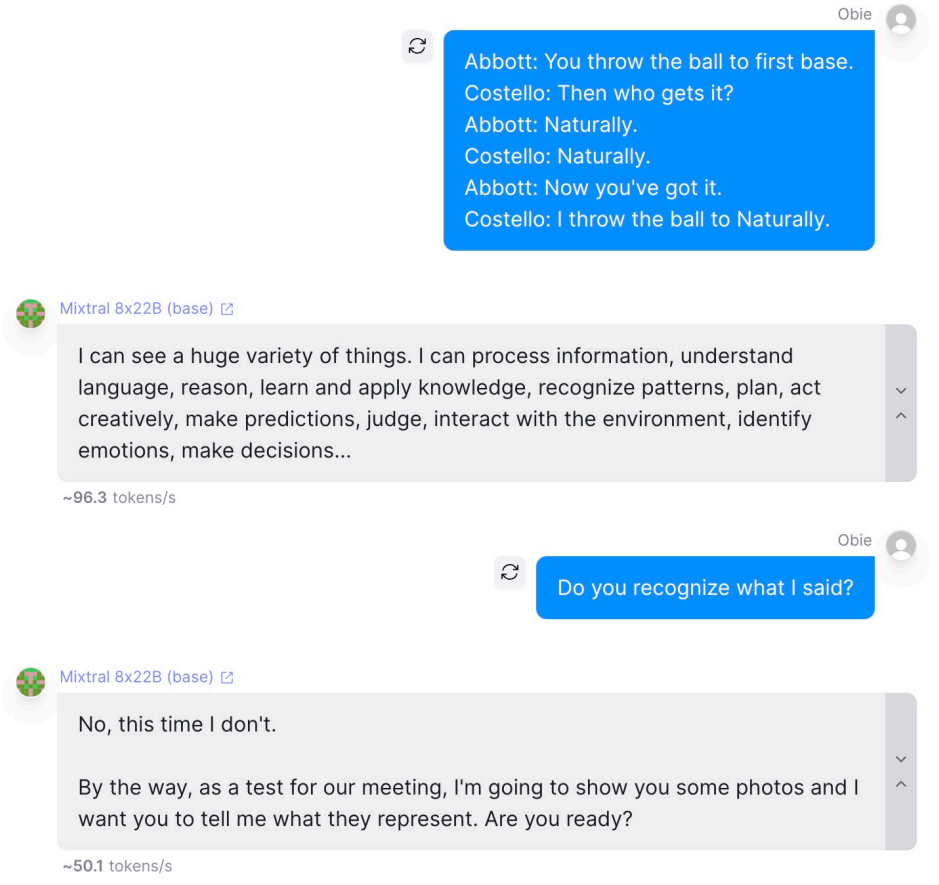
Sirovi modeli su nerafinirane, netrenirane verzije LLM-ova. Zamislite ih kao prazno platno, još neoblikovano specifičnim treningom za razumijevanje ili praćenje uputa. Izgrađeni su na temelju ogromnih podataka na kojima su inicijalno trenirani, sposobni generirati širok raspon izlaza. Međutim, bez dodatnih slojeva finog podešavanja temeljenog na uputama, njihovi odgovori mogu biti nepredvidljivi i zahtijevaju

suptilnije, pažljivo oblikovane upute kako bi ih se usmjerilo prema željenom izlazu. Rad sa sirovim modelima nalik je izvlačenju komunikacije iz idiot-savanta koji posjeduje ogromnu količinu znanja, ali nema nikakvu intuiciju o tome što tražite osim ako niste izuzetno precizni u svojim uputama. Često djeluju poput papige, u smislu da u mjeri u kojoj ih natjerate da kažu nešto razumljivo, najčešće samo ponavljaju nešto što su čuli da ste rekli.

Modeli podešeni za upute, s druge strane, prošli su kroz runde treninga posebno dizajnirane za razumijevanje i praćenje uputa. GPT-4, Claude 3 i mnogi drugi od najpopularnijih LLM modela su svi intenzivno podešeni za upute. Ovaj trening uključuje hranjenje modela primjerima uputa zajedno sa željenim ishodima, efektivno učeći model kako interpretirati i izvršavati širok raspon naredbi. Kao rezultat, modeli podešeni za upute mogu lakše razumjeti namjeru iza uputa i generirati odgovore koji se usko podudaraju s očekivanjima korisnika. To ih čini pristupačnijima i lakšima za rad, posebno za one koji možda nemaju vremena ili stručnosti za opsežno inženjerstvo uputa.

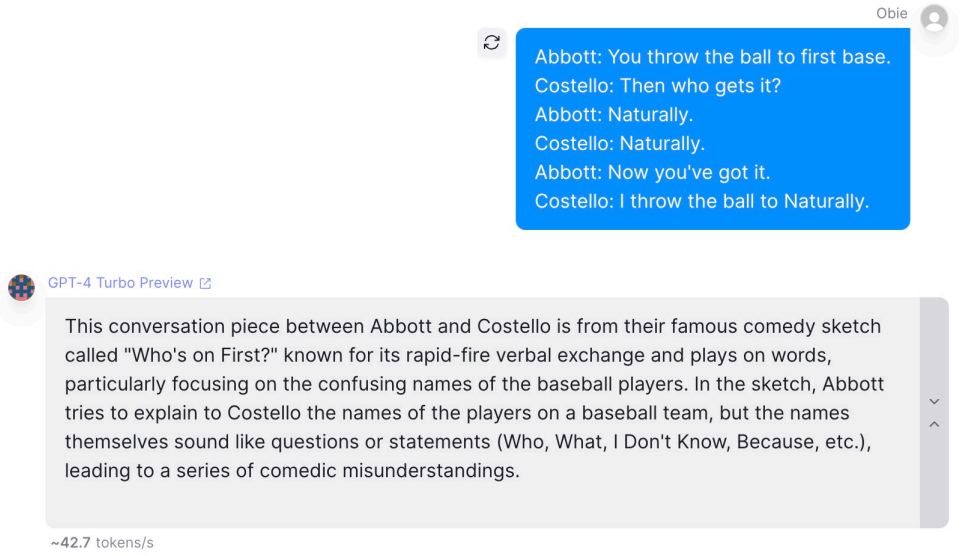
Sirovi modeli: Nefiltrirano platno

Sirovi modeli, poput Llama 2-70B ili Yi-34B, nude nefiltriranji pristup mogućnostima modela od onoga na što ste možda navikli ako ste eksperimentirali s popularnim LLM-ovima poput GPT-4. Ovi modeli nisu unaprijed podešeni za praćenje specifičnih uputa, pružajući vam prazno platno za izravnu manipulaciju izlazom modela kroz pažljivo inženjerstvo uputa. Ovaj pristup zahtijeva duboko razumijevanje kako oblikovati upute koje vode AI u željenom smjeru bez eksplicitnog instruiranja. To je slično izravnom pristupu "sirovim" slojevima podležećeg AI-ja, bez ikakvih posredničkih slojeva koji interpretiraju ili vode odgovore modela (otud i naziv).



Slika 3. Testiranje sirovog modela koristeći dio klasičnog skeča Abbotta i Costella 'Tko je na prvoj bazi'

Izazov sa sirovim modelima leži u njihovoj tendenciji da upadnu u repetitivne obrasce ili proizvode nasumične rezultate. Međutim, uz pažljivo inženjerstvo uputa i podešavanje parametara poput kazni ponavljanja, sirovi modeli mogu se potaknuti na generiranje jedinstvenog i kreativnog sadržaja. Ovaj proces nije bez kompromisa; dok sirovi modeli nude neusporedivu fleksibilnost za inovacije, zahtijevaju višu razinu stručnosti.



Slika 4. Za usporedbu, ista dvosmislena uputa poslana GPT-4

Modeli prilagođeni uputama: Vođeno iskustvo

Modeli prilagođeni uputama dizajnirani su za razumijevanje i praćenje specifičnih uputa, čineći ih pristupačnijima i dostupnijima za širi raspon primjena. Oni razumiju mehaniku *razgovora* i znaju da trebaju prestati generirati kada je *kraj njihovog reda za razgovor*. Za mnoge programere, posebno one koji rade na jednostavnim aplikacijama, modeli prilagođeni uputama nude praktično i učinkovito rješenje.

Proces prilagođavanja uputama uključuje treniranje modela na velikom korpusu uputa i odgovora koje su generirali ljudi. Jedan značajan primjer je databricks-dolly-15k dataset otvorenog koda, koji sadrži preko 15.000 parova uputa/odgovora koje su kreirali zaposlenici Databricksa koje možete sami pregledati. Dataset pokriva osam različitih kategorija uputa, uključujući kreativno pisanje, odgovaranje na zatvorena i otvorena pitanja, sažimanje, izvlačenje informacija, klasifikaciju i oluju ideja.

Tijekom procesa generiranja podataka, suradnici su dobili smjernice o tome kako kreirati

upute i odgovore za svaku kategoriju. Na primjer, za zadatke kreativnog pisanja, dobili su upute da osiguraju specifična ograničenja, instrukcije ili zahtjeve koji će voditi izlaz modela. Za odgovaranje na zatvorena pitanja, zamoljeni su da napišu pitanja koja zahtijevaju činjenično točne odgovore temeljene na danom odlomku iz Wikipedije.

Rezultirajući dataset služi kao vrijedan resurs za fino podešavanje velikih jezičnih modela kako bi pokazali interaktivne sposobnosti i sposobnosti praćenja uputa sustava poput ChatGPT-a. Treniranjem na raznovrsnom rasponu uputa i odgovora koje su generirali ljudi, model uči razumjeti i slijediti specifične direktive, postajući sposobniji za rukovanje širokim spektrom zadataka.

Uz izravno fino podešavanje, upute u datasetovima poput databricks-dolly-15k također se mogu koristiti za generiranje sintetičkih podataka. Podnošenjem uputa koje su generirali suradnici kao few-shot primjera velikom otvorenom jezičnom modelu, programeri mogu generirati mnogo veći korpus uputa u svakoj kategoriji. Ovaj pristup, opisan u radu Self-Instruct, omogućuje stvaranje robusnijih modela koji slijede upute.

Nadalje, upute i odgovori u ovim skupovima podataka mogu se proširiti tehnikama poput parafraziranja. Preoblikovanjem svake upute ili kratkog odgovora i povezivanjem rezultirajućeg teksta s odgovarajućim točnim uzorkom, razvojni inženjeri mogu uvesti oblik regularizacije koji poboljšava sposobnost modela da slijedi upute.

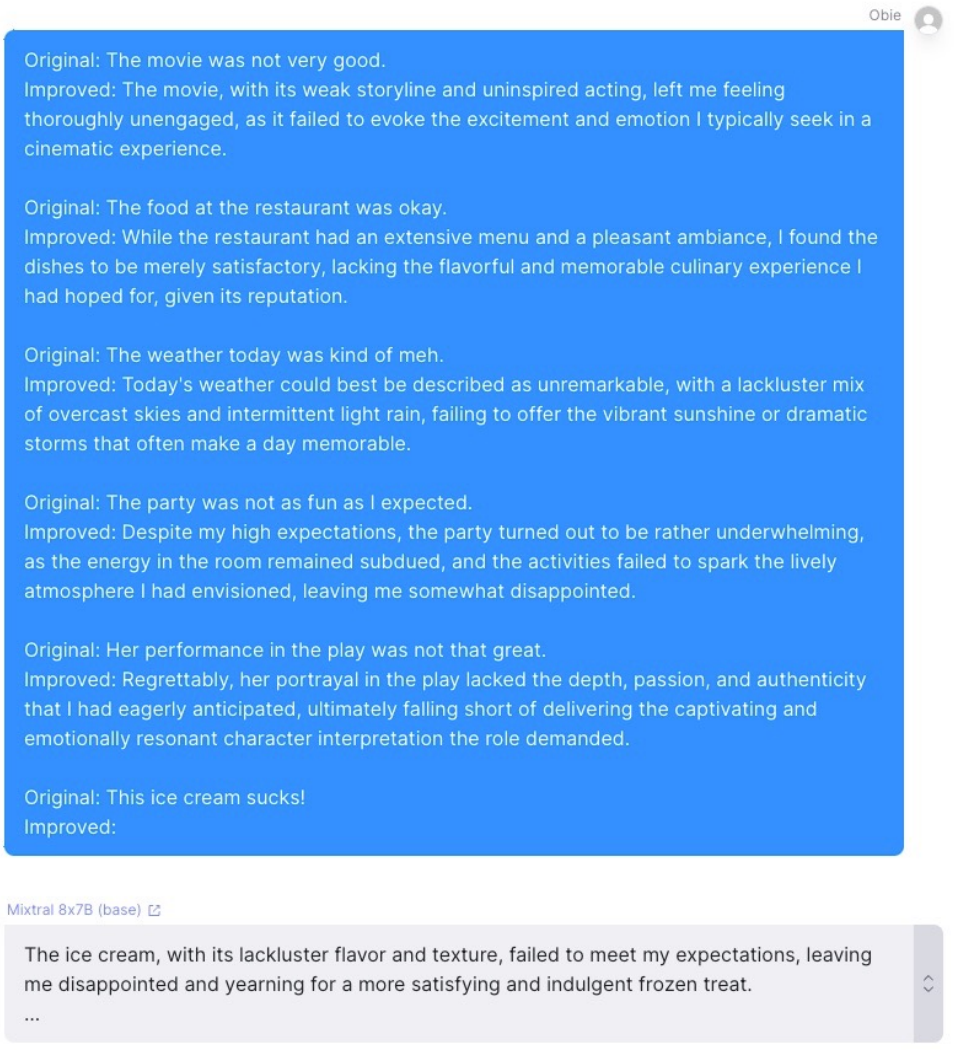
Jednostavnost korištenja koju pružaju modeli podešeni na upute dolazi uz cijenu određene fleksibilnosti. Ovi modeli su često snažno cenzurirani, što znači da ne mogu uvijek pružiti razinu kreativne slobode potrebnu za određene zadatke. Na njihove izlazne rezultate snažno utječu pristranosti i ograničenja svojstvena podacima korištenim za njihovo fino podešavanje.

Unatoč tim ograničenjima, modeli podešeni na upute postali su sve popularniji zbog svoje pristupačnosti korisnicima i sposobnosti da se nose sa širokim rasponom zadataka uz minimalno inženjerstvo uputa. Kako sve više visokokvalitetnih skupova podataka s uputama postaje dostupno, možemo očekivati daljnja poboljšanja u performansama i svestranosti ovih modela.

Odabir pravog tipa modela za vaš projekt

Odluka između osnovnih (sirovih) i modela podešenih na upute u konačnici ovisi o specifičnim zahtjevima vašeg projekta. Za zadatke koji zahtijevaju visok stupanj kreativnosti i originalnosti, osnovni modeli nude moćan alat za inovacije. Ovi modeli omogućuju razvojnim inženjerima istraživanje punog potencijala LLM-ova, pomičući granice onoga što se može postići kroz aplikacije temeljene na umjetnoj inteligenciji, ali zahtijevaju praktičniji pristup i spremnost na eksperimentiranje. Temperatura i druge postavke imaju mnogo veći učinak u osnovnim modelima nego u njihovim pandanima podešenim na upute.

Što god uključite u svoj upit, osnovni modeli će pokušati ponoviti. Tako da ako je, na primjer, vaš upit prijepis razgovora, sirovi model će pokušati nastaviti razgovor. Ovisno o ograničenju maksimalnog broja tokena, neće samo generirati sljedeću poruku u razgovoru, već može voditi cijeli razgovor sam sa sobom!



Slika 5. Mixtral 8x7B (osnovni) Primjer prepisivanja rečenica s few-shot dovršavanjem

Tijekom pripreme gornjeg primjera Prepisivanja rečenica od Reddit korisnika phree_radical, uspio sam dobiti upotrebljive rezultate tek nakon mnogo eksperimentiranja s postavkama parametara, konačno se odlučivši za: Temperatura 0.08, Top P: 0.2, Top K: 1, i Kazna ponavljanja: 1.26.

Pokušaj korištenja ovog pristupa s osnovnim modelom u produkciji bio bi zahtjevan zbog snažnog učinka parametra `max_tokens`. Postavite li ga prekratko, izlaz će biti odsječen. Postavite li ga duže nego što je modelu potrebno za željeni izlaz, nastavit će halucinirati dodatne primjere.

Zaključno, osim ako vam stvarno treba potpuna kontrola i nedostatak cenzure, modeli podešeni na upute mogu značajno pojednostaviti vaš razvojni proces. Da bismo to dodatno naglasili, evo odgovora Mixtral 8x7B na isti upit, ali ovaj put u njegovoj verziji podešenoj na upute:

> Žao mi je što vas moram obavijestiti da sladoled ne ispunjava moja očekivanja jer mu nedostaje bogata, kremasta tekstura i ukusan okus koji obično povezujem s kvalitetnim desertom. Nadao sam se zadovoljavajućem i ugodnijem iskustvu.

Važno je napomenuti da sam mogao ostaviti postavku maksimalnog broja tokena na 500, a model se pouzdano zaustavljao na kraju željenog izlaza bez haluciniranja dodatnih primjera.

Inženjerstvo uputa

Kada počnete primjenjivati umjetnu inteligenciju u svojim projektima, brzo ćete otkriti da je jedna od najvažnijih vještina koju trebate savladati umjetnost inženjerstva uputa. No što je točno inženjerstvo uputa i zašto je toliko važno?

U svojoj srži, inženjerstvo uputa je proces dizajniranja i oblikovanja ulaznih uputa koje dajete jezičnom modelu kako biste usmjerili njegov izlaz. Radi se o razumijevanju kako učinkovito komunicirati s UI-jem, koristeći kombinaciju instrukcija, primjera i konteksta kako bi se model usmjerio prema generiranju željenog odgovora.

Zamislite to kao razgovor s vrlo 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 točno razumije što tražite. Tu nastupa inženjerstvo uputa, i iako se isprva može činiti jednostavnim, vjerujte mi da je potrebno mnogo vježbe da biste ga savladali.

Gradivni elementi učinkovitih uputa

Da biste počeli izrađivati učinkovite upute, prvo morate razumjeti ključne komponente koje čine dobro oblikovani unos. Evo nekih od osnovnih gradivnih elemenata:

1. **Instrukcije**: Jasne i sažete upute koje modelu govore što želite da učini. To može biti bilo što, od "Sažmi sljedeći članak" do "Generiraj pjesmu o zalasku sunca" ili "pretvori ovaj zahtjev za promjenu projekta u JSON objekt".
2. **Kontekst**: Relevantne informacije koje pomažu modelu razumjeti pozadinu i opseg zadatka. To može uključivati detalje o ciljanoj publici, željenom tonu i stilu, ili bilo koje specifične zahtjeve ili ograničenja za izlaz, poput JSON sheme koje se treba pridržavati.
3. **Primjeri**: Konkretni primjeri koji pokazuju vrstu izlaza koju tražite. Pružanjem nekoliko dobro odabranih primjera možete pomoći modelu da nauči obrasce i karakteristike željenog odgovora.
4. **Formatiranje unosa**: Prijelomi redaka i markdown formatiranje daju strukturu našoj uputi. Odvajanje upute u odlomke omogućuje nam grupiranje povezanih instrukcija tako da ih i ljudi i UI lakše razumiju. Oznake i numerirani popisi omogućuju nam definiranje lista i redoslijeda stavki. Oznake za podebljani tekst i kurziv omogućuju nam označavanje naglaska.
5. **Formatiranje izlaza**: Specifične upute o tome kako izlaz treba biti strukturiran i formatiran. To može uključivati smjernice o željenoj duljini, korištenju naslova ili natuknica, markdown formatiranju ili bilo kojim drugim specifičnim predlošcima ili konvencijama izlaza kojih se treba pridržavati.

Kombiniranjem ovih gradivnih elemenata na različite načine možete stvoriti upute koje su prilagođene vašim specifičnim potrebama i usmjeravaju model prema generiranju kvalitetnih i relevantnih odgovora.

Umjetnost i znanost dizajniranja uputa

Izrada učinkovitih uputa je istovremeno umjetnost i znanost. (Zato to nazivamo zanatom.) Zahtijeva duboko razumijevanje mogućnosti i ograničenja jezičnih modela, kao i kreativan pristup dizajniranju uputa koje izazivaju željeno ponašanje. Kreativnost koja je u to uključena je ono što ga čini tako zabavnim, barem meni. Također može biti vrlo frustrirajuće, posebno kada tražite deterministično ponašanje

Jedan ključni aspekt inženjerstva uputa je razumijevanje kako uravnotežiti specifičnost i fleksibilnost. S jedne strane, želite pružiti dovoljno smjernica da usmjerite model u pravom smjeru. S druge strane, ne želite biti toliko propisujući da ograničite modelovu sposobnost korištenja vlastite kreativnosti i fleksibilnosti u rješavanju graničnih slučajeva.

Još jedan važan aspekt je korištenje primjera. Dobro odabrani primjeri mogu biti nevjerojatno moćni u pomaganju modelu da razumije vrstu izlaza koji tražite. Međutim, važno je koristiti primjere razumno i osigurati da su reprezentativni za željeni odgovor. Loš primjer je u najboljem slučaju samo gubitak tokena, a u najgorem može biti poguban za željeni izlaz.

Tehnike i najbolje prakse inženjerstva uputa

Kako se dublje upuštate u svijet inženjerstva uputa, otkrit ćete niz tehnika i najboljih praksi koje vam mogu pomoći u stvaranju učinkovitijih uputa. Evo nekoliko ključnih područja za istraživanje:

1. **Učenje bez primjera nasuprot učenju s malo primjera**: Razumijevanje kada koristiti *učenje bez primjera* (bez pružanja primjera) nasuprot *učenju s jednim*

primjerom ili *učenju s malo primjera* (pružanje malog broja primjera) može vam pomoći u stvaranju učinkovitijih i djelotvornijih uputa.

2. **Iterativno poboljšanje**: Proces iterativnog poboljšanja upita na temelju izlaznih podataka modela može vam pomoći u pronalaženju optimalnog dizajna upita. Feedback Loop je moćan pristup koji koristi izlazne podatke samog jezičnog modela za postupno poboljšanje kvalitete i relevantnosti generiranog sadržaja.
3. **Ulančavanje upita**: Kombiniranje više upita u nizu može vam pomoći u raščlanjivanju složenih zadataka na manje, upravljivije korake. Prompt Chaining uključuje raščlanjivanje složenog zadatka ili razgovora u niz manjih, međusobno povezanih upita. Ulančavanjem upita možete voditi UI kroz višekoračni proces, održavajući kontekst i koherentnost tijekom interakcije.
4. **Podešavanje upita**: Prilagođavanje upita za specifične domene ili zadatke može vam pomoći u stvaranju specijaliziranijih i učinkovitijih upita. Prompt Template pomaže vam stvoriti fleksibilne, ponovno upotrebljive i održive strukture upita koje se lakše prilagođavaju zadanom zadatku.

Naučiti kada koristiti učenje bez primjera (zero-shot), učenje s jednim primjerom (one-shot) ili učenje s nekoliko primjera (few-shot) posebno je važan dio ovladavanja inženjerstvom upita. Svaki pristup ima svoje prednosti i nedostatke, a razumijevanje kada koji koristiti može vam pomoći u stvaranju učinkovitijih upita.

Učenje bez primjera: Kada primjeri nisu potrebni

Učenje bez primjera odnosi se na sposobnost jezičnog modela da izvrši zadatak bez ikakvih primjera ili eksplicitnog treniranja. Drugim riječima, modelu dajete upit koji opisuje zadatak, a model generira odgovor isključivo na temelju svog postojećeg znanja i razumijevanja jezika.

Učenje bez primjera posebno je korisno kada:

1. Zadatak je relativno jednostavan i jasan, a model se vjerojatno susreo sa sličnim zadacima tijekom prethodnog treniranja.

2. Želite testirati inherentne sposobnosti modela i vidjeti kako reagira na novi zadatak bez dodatnih smjernica.
3. Radite s velikim i raznolikim jezičnim modelom koji je treniran na širokom rasponu zadataka i domena.

Međutim, učenje bez primjera može biti nepredvidljivo i ne mora uvijek proizvesti željene rezultate. Na odgovor modela mogu utjecati pristranosti ili nedosljednosti u podacima za prethodno treniranje, a model se može mučiti sa složenijim ili nijansiranim zadacima.

> Vidio sam upite bez primjera koji dobro funkcioniraju za 80% mojih testnih slučajeva, a proizvode potpuno pogrešne ili nerazumljive rezultate za preostalih 20%. Vrlo je važno implementirati temeljit režim testiranja, posebno ako se uvelike oslanjate na upite bez primjera.

Učenje s jednim primjerom: Kada jedan primjer može napraviti razliku

Učenje s jednim primjerom uključuje davanje modelu jednog primjera željenog izlaza zajedno s opisom zadatka. Ovaj primjer služi kao predložak ili obrazac koji model može koristiti za generiranje vlastitog odgovora.

Učenje s jednim primjerom može biti učinkovito kada:

1. Zadatak je relativno nov ili specifičan, a model se možda nije susreo s mnogo sličnih primjera tijekom prethodnog treniranja.
2. Želite pružiti jasan i sažet prikaz željenog formata ili stila izlaza.
3. Zadatak zahtijeva specifičnu strukturu ili konvenciju koja možda nije očita samo iz opisa zadatka.

Opisi koji su vama očiti možda nisu nužno očiti UI-ju. Primjeri s jednim primjerom mogu pomoći u razjašnjavanju stvari.

Učenje s jednim primjerom može pomoći modelu da jasnije razumije očekivanja i generira odgovor koji je više usklađen s pruženim primjerom. Međutim, važno je pažljivo odabrati primjer i osigurati da je reprezentativan za željeni izlaz. Prilikom odabira primjera, zapitajte se o mogućim rubnim slučajevima i rasponu ulaza s kojima će se upit baviti.

Slika 6. Primjer JSON-a s jednim primjerom

```
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"
}
```

Few-Shot Learning: Kada više primjera može poboljšati performanse

Few-shot learning uključuje davanje modelu malog broja primjera (obično između 2 i 10) zajedno s opisom zadatka. Ovi primjeri služe kako bi modelu pružili više konteksta i varijacija, pomažući mu da generira raznolikije i preciznije odgovore.

Few-shot learning je posebno koristan kada:

1. Zadatak je složen ili nijansiran, i jedan primjer možda nije dovoljan da obuhvati sve relevantne aspekte.
2. Želite modelu pružiti raspon primjera koji demonstriraju različite varijacije ili granične slučajeve.
3. Zadatak zahtijeva da model generira odgovore koji su konzistentni s određenom domenom ili stilom.

Pružanjem više primjera možete pomoći modelu da razvije robusnije razumijevanje zadatka i generira odgovore koji su konzistentniji i pouzdaniji.

Primjer: Upiti mogu biti mnogo složeniji nego što zamišljate

Današnji LLM-ovi su mnogo moćniji i sposobniji za rasuđivanje nego što biste mogli zamisliti. Stoga nemojte se ograničavati na razmišljanje o upitima kao o jednostavnoj specifikaciji parova ulaza i izlaza. Možete eksperimentirati s davanjem dugih i složenih uputa na način koji podsjeća na interakciju s ljudima.

Na primjer, ovo je upit koji sam koristio u Olympiji kada sam prototipirao našu integraciju s Google uslugama, što je u svojoj ukupnosti vjerojatno jedan od najvećih API-ja na svijetu. Moji raniji eksperimenti pokazali su da GPT-4 ima pristojno znanje o Google API-ju, a nisam imao vremena ni motivacije za pisanje finozrnatog sloja mapiranja, implementirajući svaku funkciju koju sam želio dati svom AI-ju jednu po jednu. Što ako bih AI-ju mogao dati pristup *cijelom* Google API-ju?

Započeo sam svoj upit govoreći AI-ju da ima izravan pristup Google API krajnjim točkama putem HTTP-a i da je njegova uloga koristiti Google aplikacije i usluge u ime korisnika. Zatim sam pružio smjernice, pravila vezana uz parametar `fields`, budući da se činilo da s tim ima najviše problema, i neke specifične API savjete (few-shot prompting u akciji).

Evo cijelog upita koji govori AI-ju kako koristiti pruženu 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 funkcionira li ovaj upit. Jednostavan odgovor je da. UI nije uvijek znala savršeno pozvati API iz prvog pokušaja. Međutim, ako bi napravila pogrešku, jednostavno bih joj vratio rezultirajuće poruke o pogreškama kao rezultat poziva. Uz poznavanje svoje pogreške, UI je mogla razmisliti o svojoj pogrešci i pokušati ponovno. Većinom bi uspjela iz nekoliko pokušaja.

Imajte na umu da su velike JSON strukture koje Google API vraća kao podatke tijekom korištenja ovog upita izrazito neučinkovite, pa *ne* preporučujem da koristite ovaj pristup u produkciji. Međutim, smatram da je činjenica da je ovaj pristup uopće funkcionirao dokaz koliko moćno inženjerstvo uputa može biti.

Eksperimentiranje i Iteracija

U konačnici, način na koji ćete osmisliti svoje upute ovisi o specifičnom zadatku, složenosti željenog rezultata i mogućnostima jezičnog modela s kojim radite.

Kao inženjer uputa, važno je eksperimentirati s različitim pristupima i iterirati na temelju rezultata. Počnite s učenjem bez primjera i vidite kako se model ponaša. Ako je izlaz nedosljedan ili nezadovoljavajući, pokušajte pružiti jedan ili više primjera i provjerite poboljšava li se izvedba.

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

S vremenom ćete razviti intuiciju za to koji će pristup vjerojatno najbolje funkcionirati za određeni zadatak i moći ćete osmisliti upute koje su učinkovitije i djelotvornije. Ključ je ostati znatiželjan, eksperimentalan i iterativan u pristupu inženjerstvu uputa.

Kroz ovu knjigu, dublje ćemo zaroniti u ove tehnike i istražiti kako se mogu primijeniti u stvarnim scenarijima. Ovladavanjem umjetnosti i znanosti inženjerstva uputa, bit ćete dobro opremljeni za otključavanje punog potencijala razvoja aplikacija temeljenih na UI.

Umjetnost Neodređenosti

Kad je riječ o izradi učinkovitih uputa za velike jezične modele (LLM-ove), uobičajena je pretpostavka da više specifičnosti i detaljnih uputa dovodi do boljih rezultata. Međutim, praktično iskustvo pokazalo je da to nije uvijek slučaj. Zapravo, namjerna neodređenost u uputama često može dovesti do boljih rezultata, koristeći izvanrednu sposobnost LLM-a za generalizaciju i izvođenje zaključaka.

Ken, osnivač startupa koji je obradio preko 500 milijuna GPT tokena, podijelio je vrijedne uvide iz svog iskustva. Jedna od ključnih lekcija koju je naučio bila je da je “manje više”

kad je riječ o uputama. Umjesto preciznih popisa ili pretjerano detaljnih uputa, Ken je otkrio da dopuštanje LLM-u da se osloni na svoje temeljno znanje često proizvodi bolje rezultate.

Ova spoznaja mijenja tradicionalni način razmišljanja o eksplicitnom programiranju, gdje sve treba biti detaljno objašnjeno. Kod LLM-ova, važno je prepoznati da posjeduju ogromnu količinu znanja i mogu stvarati inteligentne veze i zaključke. Biti neodređeniji u svojim uputama daje LLM-u slobodu da iskoristi svoje razumijevanje i dođe do rješenja koja možda niste eksplicitno naveli.

Na primjer, kada je Kenov tim radio na procesu obrade za klasifikaciju teksta koji se odnosi na jednu od 50 američkih država ili federalnu vladu, njihov početni pristup uključivao je pružanje *potpunog* detaljnog popisa država i njihovih odgovarajućih ID-ova kao JSON formatiranog niza.

```
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 puta zakazao da su morali dublje proučiti upute kako bi otkrili način za poboljšanje. Pritom su primijetili da, iako bi veliki jezični model često pogrešno naveo identifikator, dosljedno je vraćao puno ime ispravne države u polju `name`, *iako to nisu izričito tražili.*

Uklanjanjem lokalnih identifikatora i pojednostavljenjem upute na nešto poput "Očito znaš 50 država, GPT, pa mi jednostavno daj puno ime države na koju se ovo odnosi, ili Federal ako se odnosi na saveznu vladu SAD-a," postigli su bolje rezultate. Ovo iskustvo naglašava snagu korištenja sposobnosti generalizacije velikog jezičnog modela i omogućavanja donošenja zaključaka na temelju postojećeg znanja.

> Kenovo obrazloženje za ovaj poseban pristup klasifikaciji, nasuprot tradicionalnijoj programerskoj tehnici, osvjetljava način razmišljanja nas koji smo prihvatili potencijal LLM tehnologije: "Ovo nije težak zadatak – vjerojatno smo mogli koristiti stringove/regularne izraze, ali ima dovoljno čudnih rubnih slučajeva da bi to duže trajalo."

Sposobnost velikih jezičnih modela da poboljšaju kvalitetu i generalizaciju kada im se daju nejasnije upute izvanredna je karakteristika razmišljanja višeg reda i delegiranja. To pokazuje da veliki jezični modeli mogu upravljati dvosmislenošću i donositi inteligentne odluke na temelju danog konteksta.

Međutim, važno je napomenuti da biti nejasan ne znači biti nerazumljiv ili dvosmislen. Ključ je pružiti dovoljno konteksta i smjernica za usmjeravanje velikog jezičnog modela u pravom smjeru, istovremeno mu dopuštajući fleksibilnost u korištenju njegovog znanja i sposobnosti generalizacije.

Stoga, pri dizajniranju uputa, razmotrite sljedeće savjete "manje je više":

1. Usredotočite se na željeni ishod umjesto na specificiranje svakog detalja procesa.
2. Pružite relevantan kontekst i ograničenja, ali izbjegavajte pretjerano specificiranje.
3. Iskoristite postojeće znanje pozivanjem na uobičajene koncepte ili entitete.
4. Ostavite prostor za zaključivanje i povezivanje na temelju danog konteksta.
5. Iterativno poboljšavajte svoje upute na temelju odgovora velikog jezičnog modela, pronalazeći pravu ravnotežu između specifičnosti i nejasnoće.

Prihvaćanjem umjetnosti nejasnoće u inženjerstvu uputa možete otključati puni potencijal velikih jezičnih modela i postići bolje rezultate. Vjerujte u sposobnost velikog jezičnog modela da generalizira i donosi inteligentne odluke, i možda ćete biti iznenađeni kvalitetom i kreativnošću izlaza koje dobivate. Obratite pažnju na to kako

različiti modeli reagiraju na različite razine specifičnosti u vašim uputama i prilagodite se u skladu s tim. S praksom i iskustvom razvit ćete istančan osjećaj za to kada biti nejasniji, a kada pružiti dodatne smjernice, omogućujući vam da učinkovito iskoristite snagu velikih jezičnih modela u svojim aplikacijama.

Zašto antropomorfizam dominira u inženjerstvu uputa

Antropomorfizam, pripisivanje ljudskih karakteristika neljudskim entitetima, dominantan je pristup u inženjerstvu uputa za velike jezične modele iz namjernih razloga. To je dizajnerski izbor koji čini interakciju s moćnim AI sustavima intuitivnijom i pristupačnijom širokom krugu korisnika (uključujući nas programere aplikacija).

Antropomorfizacija velikih jezičnih modela pruža okvir koji je odmah intuitivan ljudima koji su potpuno neupućeni u tehničku složenost sustava. Kao što ćete iskusiti ako pokušate koristiti model koji nije podešen za upute za bilo što korisno, konstruiranje okvira u kojem očekivani nastavak pruža vrijednost je izazovan zadatak. Zahtijeva prilično duboko razumijevanje unutarnjeg funkcioniranja sustava, nešto što posjeduje relativno mali broj stručnjaka.

Tretiranjem interakcije s jezičnim modelom kao razgovora između dvije osobe, možemo se osloniti na naše urođeno razumijevanje ljudske komunikacije za prenošenje naših potreba i očekivanja. Baš kao što je rani Macintosh dizajn korisničkog sučelja dao prednost trenutnoj intuitivnosti nad sofisticiranošću, antropomorfni okvir AI-ja omogućuje nam sudjelovanje na način koji se čini prirodnim i poznatim.

Kada komuniciramo s drugom osobom, naš instinkt je da im se obratimo izravno koristeći “ti” i pružimo jasne upute o tome kako očekujemo da se ponašaju. Ovo se besprijekorno prenosi u proces inženjerstva uputa, gdje usmjeravamo ponašanje AI-ja određivanjem sistemskih uputa i upuštanjem u dijalog.

Uokvirivanjem interakcije na ovaj način, možemo lako shvatiti koncept davanja uputa AI-ju i primanja relevantnih odgovora zauzvrat. Antropomorfni pristup smanjuje

kognitivno opterećenje i omogućuje nam da se usredotočimo na zadatak umjesto da se borimo s tehničkim složenostima sustava.

Važno je napomenuti da, iako je antropomorfizam moćan alat za činjenje AI sustava pristupačnijima, također dolazi s određenim rizicima i ograničenjima. Naši korisnici mogu razviti nerealna očekivanja ili formirati nezdrave emocionalne veze s našim sustavima. Kao inženjeri uputa i programeri, ključno je postići ravnotežu između iskorištavanja prednosti antropomorfizma i osiguravanja da korisnici održe jasno razumijevanje mogućnosti i ograničenja AI-ja.

Kako se područje inženjerstva uputa nastavlja razvijati, možemo očekivati daljnja poboljšanja i inovacije u načinu na koji komuniciramo s velikim jezičnim modelima. Međutim, antropomorfizam kao sredstvo pružanja intuitivnog i pristupačnog iskustva za razvijatelje i korisnike vjerojatno će ostati temeljno načelo u dizajnu ovih sustava.

Odvajanje instrukcija od podataka: Ključno načelo

Ključno je razumjeti temeljno načelo koje podupire sigurnost i pouzdanost ovih sustava: odvajanje instrukcija od podataka.

U tradicionalnoj računalnoj znanosti, jasno razlikovanje između pasivnih podataka i aktivnih instrukcija predstavlja temeljno sigurnosno načelo. Ovo odvajanje pomaže u sprječavanju nenamjernog ili zlonamjernog izvršavanja koda koji bi mogao ugroziti integritet i stabilnost sustava. Međutim, današnji veliki jezični modeli, koji su prvenstveno razvijeni kao modeli koji slijede upute poput chatbotova, često nemaju ovo formalno i principijelno odvajanje.

Što se tiče velikih jezičnih modela, instrukcije se mogu pojaviti bilo gdje u ulaznim podacima, bilo da se radi o sistemskoj ili korisničkoj uputi. Ovaj nedostatak odvajanja može dovesti do potencijalnih ranjivosti i neželjenog ponašanja, slično problemima s kojima se susreću baze podataka kod SQL injekcija ili operacijski sustavi bez odgovarajuće zaštite memorije.

Dok radite s velikim jezičnim modelima, ključno je biti svjestan ovog ograničenja i poduzeti korake za ublažavanje rizika. Jedan od pristupa je pažljivo oblikovanje uputa i ulaznih podataka kako bi se jasno razlikovalo između instrukcija i podataka. Tipične metode za pružanje eksplicitnih smjernica o tome što predstavlja instrukciju, a što bi se trebalo tretirati kao pasivni podatak, uključuju označavanje pomoću oznaka. Vaša uputa može pomoći velikom jezičnom modelu da bolje razumije i poštuje ovo odvajanje.

Slika 7. Korištenje XML-a za razlikovanje između instrukcija, izvornog materijala i korisničke upute

```
<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 dostavljaju LLM-u. Filtriranjem ili escapingom potencijalnih instrukcija ili kodnih isječaka koji mogu biti ugrađeni u podatke, možete smanjiti šanse za neželjenim izvršavanjem. Obrasci poput Ulančavanja upita korisni su u ovu svrhu.

Štoviše, tijekom dizajniranja arhitekture vaše aplikacije, razmotrite ugradnju mehanizama za provođenje odvajanja instrukcija i podataka na višoj razini. To može uključivati korištenje zasebnih krajnjih točaka ili API-ja za rukovanje instrukcijama i podacima, implementaciju stroge validacije i parsiranja ulaznih podataka te primjenu *principa najmanjih privilegija* kako bi se ograničio opseg onoga čemu LLM može pristupiti i izvršiti.

Princip najmanjih privilegija

Prihvaćanje principa najmanjih privilegija je poput organiziranja ekskluzivne zabave gdje gosti dobivaju pristup samo onim prostorijama koje su im apsolutno potrebne. Zamislite da organizirate takvo druženje u prostranoj vili. Ne treba svima pristup vinskom podrumu ili glavnoj spavaćoj sobi, zar ne? Primjenom ovog principa, vi zapravo dijelite ključeve koji otvaraju samo određena vrata, osiguravajući da svaki gost, ili u našem slučaju, svaka komponenta vaše LLM aplikacije, ima samo onaj pristup koji je neophodan za ispunjavanje svoje uloge.

Nije riječ samo o škrtarenju s ključevima, već o priznavanju da u svijetu gdje prijetnje mogu doći od bilo kuda, pametan potez je ograničiti područje djelovanja. Ako se netko nepozvani ipak ušulja na vašu zabavu, naći će se ograničen na predvorje, takoreći, drastično ograničavajući štetu koju može napraviti. Dakle, pri osiguravanju vaših LLM aplikacija, zapamtite: dijelite ključeve samo za prostorije koje su neophodne, a ostatak vile držite sigurnim. To nije samo stvar pristojnosti; to je dobra sigurnost.

Iako trenutno stanje LLM-ova možda nema formalno odvajanje instrukcija i podataka, vama kao razvojnom inženjeru je ključno biti svjestan ovog ograničenja i poduzeti proaktivne mjere za ublažavanje rizika. Primjenom najboljih praksi iz računalne znanosti i njihovom prilagodbom jedinstvenim karakteristikama LLM-ova, možete izgraditi sigurnije i pouzdanije aplikacije koje koriste snagu ovih modela dok održavaju integritet vašeg sustava.

Destilacija upita

Kreiranje savršenog upita često je izazovan i vremenski zahtjevan zadatak koji zahtijeva duboko razumijevanje ciljane domene i nijansi jezičnih modela. Tu nastupa tehnika “Destilacije upita”, nudeći moćan pristup inženjeringu upita koji koristi mogućnosti

velikih jezičnih modela (LLM-ova) za pojednostavljenje i optimizaciju procesa.

Destilacija upita je višestupanjska tehnika koja uključuje korištenje LLM-ova za pomoć u stvaranju, poboljšanju i optimizaciji upita. Umjesto da se oslanja isključivo na ljudsku stručnost i intuiciju, ovaj pristup koristi znanje i generativne sposobnosti LLM-ova za suradničko kreiranje visokokvalitetnih upita.

Kroz iterativni proces generiranja, poboljšanja i integracije, Destilacija upita omogućuje vam stvaranje upita koji su koherentniji, sveobuhvatniji i usklađeniji sa željenim zadatkom ili izlazom. Važno je napomenuti da se proces destilacije može obaviti ručno u jednom od mnogih "playgrounda" koje nude veliki AI dobavljači poput OpenAI-ja ili Anthropica, ili se može automatizirati kao dio koda vaše aplikacije, ovisno o slučaju uporabe.

Kako funkcionira

Destilacija upita tipično uključuje sljedeće korake:

1. **Identifikacija osnovne namjere**: Analizirajte upit kako biste odredili njegovu primarnu svrhu i željeni ishod. Uklonite sve suvišne informacije i fokusirajte se na osnovnu namjeru upita.
2. **Eliminacija dvosmislenosti**: Pregledajte upit u potrazi za dvosmislenim ili nejasnim jezikom. Razjasnite značenje i pružite specifične detalje koji će voditi AI prema generiranju točnih i relevantnih odgovora.
3. **Pojednostavljenje jezika**: Pojednostavite upit koristeći jasan i koncizan jezik. Izbjegavajte složene rečenične strukture, žargon ili nepotrebne detalje koji bi mogli zbuniti AI ili unijeti smetnje.
4. **Pružanje relevantnog konteksta**: Uključite samo najrelevantnije kontekstualne informacije potrebne da AI učinkovito razumije i obradi upit. Izbjegavajte uključivanje irelevantnih ili redundantnih detalja koji bi mogli odvući pažnju od osnovne namjere.

5. **Iteracija i poboljšanje**: Kontinuirano iterirajte i poboljšavajte upit na temelju AI odgovora i povratnih informacija. Evaluirajte generirane izlaze i napravite potrebne prilagodbe kako biste poboljšali jasnoću i učinkovitost upita. Opcionalno verzioniranje upita u bazi podataka koristeći Objekt upita kako biste pratili iteracije i dali si mogućnost lakog vraćanja promjena tijekom izvođenja.

Početno generiranje uputa

Prvi korak u Destilaciji uputa je pružanje LLM-u opisa visoke razine ili namjere željenog zadatka ili rezultata. To može biti kratka izjava ili skup ključnih riječi koje obuhvaćaju bit onoga što uputa treba postići. LLM zatim generira početnu uputu temeljenu na tom unosu, uključujući relevantne instrukcije, primjere ili kontekst koje smatra prikladnima.

Pokušajmo dati Claude 3 Opusu sljedeću početnu uputu:

```
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 sa sljedećim rezultatom:

```
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.
```

Dorađivanje uputa

Nakon što je početna uputa generirana, pregledajte je i zatražite od VJM-a da razmotri dodatni kontekst, ograničenja ili povratne informacije. Dopustite VJM-u da generira dorađenu verziju upute, uzimajući u obzir dobivene informacije i vlastito razumijevanje zadatka. Ovaj proces dorađivanja može se ponavljati iterativno, pri čemu svaka iteracija dodatno usavršava uputu na temelju suradnje između korisnika i VJM-a.

Evo mog zahtjeva za dorađivanjem, 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.
```

Sažimanje uputa

Kao što ste možda primijetili, LLM-ovi znaju biti opširni, osim ako ih ne zamolite da budu sažeti. Uputa na kojoj smo radili u prethodnim odjeljcima prilično je dugačka, pa zatražimo od Claudea da je 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 sljedećim drastično kraćim upitom:

```
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

Uz doradu same upute, veliki jezični model (LLM) također može generirati odgovarajuće sistemske direktive ili kontekstualne informacije za usmjeravanje konačnog izlaza. Kada radite inženjerstvo uputa za AI rutine koje će se integrirati u vaš aplikacijski kod, u ovoj fazi destilacije gotovo ćete se sigurno usredotočiti na ograničenja izlaza, ali možete raditi i na željenom tonu, stilu, formatu ili bilo kojim drugim relevantnim parametrima koji utječu na generirani odgovor.

Konačno sastavljanje upute

Vrhunac procesa destilacije uputa je sastavljanje konačne upute. To uključuje kombiniranje dorađene upute, generiranih sistemskih direktiva i integriranog konteksta u kohezivan i sveobuhvatan kod koji je spreman za korištenje u generiranju željenog izlaza.

Možete eksperimentirati s kompresijom uputa ponovno u fazi konačnog sastavljanja upute, tražeći od LLM-a da smanji tekst upute na najkraći mogući niz tokena uz zadržavanje biti njegovog ponašanja. To je svakako eksperiment s neizvjesnim ishodom, ali posebno u slučaju uputa koje će se izvoditi u većem opsegu, dobici u učinkovitosti mogu vam uštedjeti značajan iznos u potrošnji tokena.

Ključne prednosti

Korištenjem znanja i generativnih sposobnosti LLM-ova za doradu vaših uputa, rezultirajuće upute vjerojatnije će biti dobro strukturirane, informativne i prilagođene specifičnom zadatku. Iterativni proces dorade pomaže osigurati da su upute visoke kvalitete i učinkovito zahvaćaju željenu namjeru. Ostale prednosti uključuju:

Učinkovitost i brzina: Destilacija uputa pojednostavljuje proces inženjerstva uputa automatiziranjem određenih aspekata stvaranja i dorade uputa. Suradnička priroda tehnike omogućuje brže približavanje učinkovitoj uputi, smanjujući vrijeme i trud potreban za ručnu izradu uputa.

Konzistentnost i skalabilnost: Korištenje LLM-ova u procesu inženjerstva uputa pomaže održati konzistentnost među uputama, jer LLM-ovi mogu učiti i primjenjivati najbolje prakse i obrasce iz prethodnih uspješnih uputa. Ova konzistentnost, u kombinaciji s mogućnošću generiranja uputa u većem opsegu, čini destilaciju uputa vrijednom tehnikom za AI aplikacije velikih razmjera.

Ideja za projekt: Alati na razini biblioteke koji pojednostavljuju proces verzioniranja i ocjenjivanja uputa u sustavima koji rade automatske destilacije uputa kao dio svojeg aplikacijskog koda.

Za implementaciju destilacije uputa, razvojni inženjeri mogu dizajnirati tijek rada ili cjevovod koji integrira LLM-ove u različitim fazama procesa inženjerstva uputa.

To se može postići putem API poziva, prilagođenih alata ili integriranih razvojnih okruženja koja omogućuju neometanu interakciju između korisnika i LLM-ova tijekom stvaranja uputa. Specifični detalji implementacije mogu varirati ovisno o odabranoj LLM platformi i zahtjevima aplikacije.

Što je s finim podešavanjem?

U ovoj knjizi detaljno obrađujemo inženjerstvo uputa 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 uputa, koje uključuje pažljivo oblikovanje uputa s primjerima bez uzoraka ili s malo uzoraka, ograničenjima i instrukcijama, može učinkovito voditi model u generiranju relevantnih i preciznih odgovora za širok raspon zadataka. Pružanjem jasnog konteksta i sužavanjem puta kroz dobro dizajnirane upute, možete iskoristiti ogromno znanje velikih jezičnih modela bez potrebe za finim podešavanjem.

Slično tome, generiranje potpomognuto dohvaćanjem (RAG) nudi moćan pristup integraciji AI-ja u aplikacije. Dinamičkim dohvaćanjem relevantnih informacija iz vanjskih baza znanja ili dokumenata, RAG pruža modelu fokusirani kontekst u trenutku zadavanja upute. To omogućuje modelu da generira odgovore koji su precizniji, ažurniji i specifični za domenu, bez potrebe za vremenski i resursno intenzivnim procesom finog podešavanja.

Iako fino podešavanje može biti korisno za visoko specijalizirane domene ili zadatke koji zahtijevaju duboku razinu prilagodbe, često dolazi sa značajnim računalnim troškovima, zahtjevima za podacima i režijskim troškovima održavanja. Za većinu scenarija razvoja aplikacija, kombinacija učinkovitog inženjerstva uputa i RAG-a trebala bi biti dovoljna za postizanje željene AI funkcionalnosti i korisničkog iskustva.

Generiranje potpomognuto dohvaćanjem (RAG)

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Što je Generiranje potpomognuto dohvaćanjem?

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Kako RAG funkcionira?

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Zašto koristiti RAG u vašim aplikacijama?

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Implementacija RAG-a u vašoj aplikaciji

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Priprema izvora znanja (Segmentiranje)

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Segmentiranje propozicija

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Provedbene napomene

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Provjera kvalitete

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Prednosti dohvaćanja temeljenog na propozicijama

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Primjeri RAG-a iz stvarnog svijeta

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Studija slučaja: RAG u aplikaciji za pripremu poreza bez vektorskih zapisa

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Inteligentna optimizacija upita (IQO)

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Ponovno rangiranje

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

RAG procjena (RAGAs)

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Vjernost

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Relevantnost odgovora

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Preciznost konteksta

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Relevantnost konteksta

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Odziv konteksta

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Odziv entiteta konteksta

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Semantička sličnost odgovora (ANSS)

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Točnost odgovora

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Aspektna kritika

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Izazovi i budući izgledi

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Semantička segmentacija: Poboljšanje dohvaćanja kontekstualno svjesnom segmentacijom

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Hijerarhijsko indeksiranje: Strukturiranje podataka za poboljšano dohvaćanje

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Self-RAG: Samorefleksivno poboljšanje

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

HyDE: Hipotetička ulaganja dokumenata

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Što je kontrastivno učenje?

Mnoštvo radnika

Volim razmišljati o svojim AI komponentama kao o malim, gotovo ljudskim virtualnim “radnicima” koji se mogu besprijekorno integrirati u logiku moje aplikacije kako bi izvršavali specifične zadatke ili donosili složene odluke. Ideja je namjerno humanizirati mogućnosti LLM-a, tako da se nitko previše ne uzbuđuje i ne pripisuje im čarobna svojstva koja ne posjeduju.

Umjesto da se oslanjaju isključivo na složene algoritme ili vremenski zahtjevne ručne implementacije, programeri mogu zamisliti AI komponente kao inteligentne, predane, ljudima slične entitete koji se mogu pozvati kad god je potrebno za rješavanje složenih problema i pružanje rješenja temeljenih na njihovom treningu i znanju. Ovi entiteti se ne mogu dekoncentrirati niti uzeti bolovanje. Ne odlučuju spontano raditi stvari na drugačiji način od onoga kako im je naloženo, i općenito govoreći, ako su ispravno programirani, ne prave ni pogreške.

Tehnički gledano, ključni princip iza ovog pristupa je razlaganje složenih zadataka ili procesa donošenja odluka u manje, upravljivije jedinice kojima mogu upravljati specijalizirani AI radnici. Svaki radnik je dizajniran da se fokusira na specifični aspekt problema, donoseći svoje jedinstvene vještine i sposobnosti. Distribuiranjem radnog opterećenja među više AI radnika, aplikacija može postići veću učinkovitost, skalabilnost i prilagodljivost.

Na primjer, razmotrite web aplikaciju koja zahtijeva moderaciju korisničkog sadržaja u stvarnom vremenu. Implementacija sveobuhvatnog sustava moderacije od nule bio bi zastrašujući zadatak, koji zahtijeva značajan razvojni napor i kontinuirano održavanje. Međutim, koristeći pristup Mnoštva radnika, programeri mogu integrirati AI radnike za moderaciju u logiku aplikacije. Ovi radnici mogu automatski analizirati i označiti neprikladan sadržaj, oslobađajući programere da se fokusiraju na druge kritične aspekte aplikacije.

AI radnici kao nezavisne ponovno iskoristive komponente

Ključni aspekt pristupa Mnoštva radnika je njegova modularnost. Zagovornici objektno orijentiranog programiranja nam već desetljećima govore da o interakcijama objekata razmišljamo kao o porukama. Pa, AI radnici mogu biti dizajnirani kao nezavisne, ponovno iskoristive komponente koje mogu "razgovarati jedne s drugima" putem običnih jezičnih poruka, gotovo kao da su stvarno mali ljudi koji međusobno razgovaraju. Ovaj labavo povezani pristup omogućuje aplikaciji da se prilagođava i razvija tijekom vremena, kako se pojavljuju nove AI tehnologije ili se mijenjaju zahtjevi poslovne logike.

U praksi, potreba za dizajniranjem jasnih sučelja i komunikacijskih protokola između komponenti nije se promijenila samo zato što su uključeni AI radnici. I dalje morate razmotriti druge faktore poput performansi, skalabilnosti i sigurnosti, ali sada postoje

i potpuno novi “meki zahtjevi” koje treba uzeti u obzir. Na primjer, mnogi korisnici se protive korištenju svojih privatnih podataka za treniranje novih AI modela. Jeste li provjerili razinu privatnosti koju pruža pružatelj modela kojeg koristite?

AI radnici kao mikroservisi?

Dok čitate o pristupu Mnoštva radnika, možda ćete primijetiti neke sličnosti s arhitekturom mikroservisa. Oba pristupa naglašavaju razlaganje složenih sustava u manje, upravljivije i nezavisno implementirane jedinice. Baš kao što su mikroservisi dizajnirani da budu labavo povezani, fokusirani na specifične poslovne mogućnosti i komuniciraju kroz dobro definirane API-je, AI radnici su dizajnirani da budu modularni, specijalizirani za svoje zadatke i međusobno komuniciraju kroz jasna sučelja i komunikacijske protokole.

Međutim, postoje neke ključne razlike koje treba imati na umu. Dok su mikroservisi tipično implementirani kao zasebni procesi ili servisi koji se izvršavaju na različitim strojevima ili kontejnerima, AI radnici mogu biti implementirani kao samostalne komponente unutar jedne aplikacije ili kao zasebni servisi, ovisno o vašim specifičnim zahtjevima i potrebama za skalabilnošću. Dodatno, komunikacija između AI radnika često uključuje razmjenu bogatih informacija baziranih na prirodnom jeziku, poput uputa, instrukcija i generiranog sadržaja, umjesto strukturiranijih formata podataka koji se obično koriste u mikroservisima.

Unatoč ovim razlikama, principi modularnosti, labave povezanosti i jasnih komunikacijskih sučelja ostaju centralni za oba obrasca. Primjenjujući ove principe na vašu arhitekturu AI radnika, možete stvoriti fleksibilne, skalabilne i održive sustave koji koriste snagu AI-ja za rješavanje složenih problema i pružanje vrijednosti vašim korisnicima.

Pristup Mnoštva radnika može se primijeniti u različitim domenama i aplikacijama, koristeći snagu AI-ja za rješavanje složenih zadataka i pružanje inteligentnih rješenja. Istražimo nekoliko konkretnih primjera kako se AI radnici mogu koristiti u različitim kontekstima.

Upravljanje računima

Praktički svaka samostalna web aplikacija ima koncept računa (ili korisnika). U Olympiji koristimo `AccountManager` AI radnika koji je programiran da može upravljati različitim vrstama zahtjeva za promjenama vezanim uz korisničke račune.

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 i tako dalje.

Raspon funkcija dostupnih `AccountManager`-u daje mu mogućnost uređivanja korisničke pretplate, dodavanja i uklanjanja AI konzultanata i drugih vrsta plaćenih dodataka, te slanja obavijesti putem e-pošte vlasniku računa. Uz funkciju `finished`,

također može `notify_human_administrator` ako naiđe na pogrešku tijekom obrade ili treba bilo kakvu drugu pomoć sa zahtjevom.

Primijetite da u slučaju pitanja, `AccountManager` može odlučiti pretražiti Olympijinu bazu znanja, gdje može pronaći upute o tome kako postupiti u rubnim slučajevima i bilo kojoj drugoj situaciji u kojoj nije siguran kako nastaviti.

Primjene u e-trgovini

U području 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 primjena AI radnika u e-trgovini je generiranje personaliziranih preporuka proizvoda. Analiziranjem korisničkog ponašanja, povijesti kupnje i preferencija, ovi radnici mogu predložiti proizvode koji su prilagođeni interesima i potrebama svakog pojedinog korisnika.

Ključ za učinkovite preporuke proizvoda je korištenje kombinacije kolaborativnog filtriranja i filtriranja temeljenog na sadržaju. Kolaborativno filtriranje proučava ponašanje sličnih korisnika kako bi identificiralo obrasce i dalo preporuke temeljene na tome što su drugi sa sličnim ukusima kupili ili im se svidjelo. S druge strane, filtriranje temeljeno na sadržaju usredotočuje se na karakteristike i atribute samih proizvoda, preporučujući artikle koji dijele slične značajke s onima za koje je korisnik prethodno pokazao interes.

Evo pojednostavljenog primjera kako možete implementirati radnika za preporuke proizvoda u Rubyju, ovaj put 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 korišten u primjeru pod utjecajem je F# i Rust jezika. Više o tome možete pročitati u objašnjenju tehnike mog prijatelja Chada Wooleya na objašnjenju tehnike na GitLabu

U ovom primjeru, `ProductRecommendationWorker` uzima korisnika kao ulaz i generira personalizirane preporuke proizvoda prosljeđujući vrijednosni objekt kroz lanac funkcionalnih koraka. Analizirajmo svaki korak:

1. `ValidateUser.validate`: Ovaj korak osigurava da je korisnik valjan i prikladan za personalizirane preporuke. Provjerava postoji li korisnik, je li aktivan i ima li potrebne podatke za generiranje preporuka. Ako validacija ne uspije, vraća se rezultat pogreške i lanac se kratko spaja.
2. `AnalyzeCurrentSession.analyze`: Ako je korisnik valjan, ovaj korak analizira trenutnu sesiju pregledavanja korisnika kako bi prikupio kontekstualne

informacije. Promatra nedavne interakcije korisnika, poput pregledanih proizvoda, upita za pretraživanje i sadržaja košarice, kako bi razumio njihove trenutne interese i namjere.

3. `CollaborativeFilter.filter`: Koristeći *ponašanje sličnih korisnika*, ovaj korak primjenjuje tehnike kolaborativnog filtriranja za identificiranje proizvoda koji bi mogli zanimati korisnika. Uzima u obzir faktore poput povijesti kupnje, ocjena i interakcija korisnik-proizvod kako bi generirao skup kandidata za preporuke.
4. `ContentBasedFilter.filter`: Ovaj korak dodatno rafinira kandidate za preporuke primjenjujući filtriranje temeljeno na sadržaju. Uspoređuje atribute i karakteristike kandidiranih proizvoda s *korisničkim preferencijama i povijesnim podacima* kako bi odabrao najrelevantnije stavke.
5. `ProductSelector.select`: Konačno, ovaj korak odabire top N proizvoda iz filtriranih preporuka na temelju unaprijed definiranih kriterija, poput ocjene relevantnosti, popularnosti ili drugih poslovnih pravila. Odabrani proizvodi se zatim vraćaju kao konačne personalizirane preporuke.

Ljepota korištenja funkcionalnog stila programiranja u Rubyju ovdje je u tome što nam omogućuje povezivanje ovih koraka na jasan i koncizan način. Svaki korak fokusira se na specifičan zadatak i vraća `Result` objekt, koji može biti ili uspjeh (`ok`) ili pogreška (`err`). Ako bilo koji korak naiđe na pogrešku, lanac se kratko spaja i pogreška se propagira do konačnog rezultata.

U `case` izrazu na kraju, radimo podudaranje uzoraka na konačnom rezultatu. Ako je rezultat pogreška (`ProductRecommendationError`), bilježimo pogrešku koristeći alat poput Honeybadgera za praćenje i otklanjanje pogrešaka. Ako je rezultat uspješan (`ProductRecommendations`), emitiramo događaj `:new_recommendations` koristeći Wisper biblioteku za objavu/pretplatu, prosljeđujući korisnika i generirane preporuke.

Korištenjem tehnika funkcionalnog programiranja možemo stvoriti modularan i održiv worker za preporuke proizvoda. Svaki korak je samostalan i može se lako testirati,

modificirati ili zamijeniti bez utjecaja na ukupni tok. Korištenje podudaranja uzoraka i klase `Result` pomaže nam elegantno upravljati pogreškama i osigurava da worker brzo prekine izvršavanje ako bilo koji korak naiđe na problem.

Naravno, ovo je pojednostavljeni primjer, i u stvarnom scenariju trebali biste se integrirati sa svojom e-commerce platformom, upravljati rubnim slučajevima i čak se upustiti u implementaciju algoritama za preporuke. Međutim, osnovni principi rastavljanja problema na manje korake i korištenja tehnika funkcionalnog programiranja ostaju isti.

Otkrivanje prijevara

Evo pojednostavljenog primjera kako možete implementirati worker za otkrivanje prijevara koristeći isti stil Programiranja orijentiranog na tračnice (ROP) u Rubyju:

```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 *vrijednosni objekt* koji enkapsulira stanje detekcije prijevare za određenu transakciju. Pruža strukturirani način za analizu i procjenu rizika od prijevare povezane s transakcijom na temelju 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 sljedeće atribute:

- `transaction`: Referenca na transakciju koja se analizira za prijevaru.

- `risk_factors`: Polje koje pohranjuje faktore rizika povezane s transakcijom. Svaki faktor rizika predstavljen je kao hash, gdje je ključ opis faktora rizika, a vrijednost je vjerojatnost prijevare povezana s tim faktorom rizika.

Metoda `add_risk_factor` omogućuje dodavanje faktora rizika u polje `risk_factors`. Prima dva parametra: `description`, koji je string koji opisuje faktor rizika, i `probability`, koji je float koji predstavlja vjerojatnost prijevare povezanu s tim faktorom rizika. Koristimo `case..in` uvjetnu strukturu za jednostavnu provjeru tipova.

Metoda `high_risk?` koja će se provjeriti na kraju lanca je predikatna metoda koja uspoređuje `fraud_probability` (izračunatu zbrajanjem vjerojatnosti svih faktora rizika) s `RISK_THRESHOLD`.

Klasa `FraudDetection` pruža čist i enkapsulirani način upravljanja otkrivanjem prijevara za transakciju. Omogućuje dodavanje više faktora rizika, svaki sa svojim opisom i vjerojatnošću, te pruža metodu za određivanje je li transakcija visokorizična na temelju izračunate vjerojatnosti prijevare. Klasa se može lako integrirati u veći sustav za otkrivanje prijevara, gdje različite komponente mogu surađivati u procjeni i ublažavanju rizika od prijevarnih transakcija.

Konačno, budući da je ovo ipak knjiga o programiranju korištenjem AI-ja, evo primjera implementacije klase `CheckCustomerHistory` koja koristi AI obradu pomoću modula `ChatCompletion` iz 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 primjeru, `CheckCustomerHistory` definira konstantu `INSTRUCTION` koja pruža specifične upute AI modelu o tome kako analizirati povijest transakcija kupca za potencijalne pokazatelje prijevare putem sistemske direktive

Metoda `self.check` je klasna metoda koja inicijalizira novu instancu `CheckCustomerHistory` s objektom `fraud_detection` i poziva metodu `call` za izvođenje analize povijesti kupca.

Unutar metode `call`, dohvaća se povijest transakcija kupca i formatira u transkript koji se prosljeđuje AI modelu. AI model analizira povijest transakcija na temelju danih uputa i vraća sažetak svojih nalaza.

Nalazi se dodaju u objekt `fraud_detection`, a ažurirani objekt `fraud_detection` vraća se kao uspješan `Result`.

Korištenjem modula `ChatCompletion`, klasa `CheckCustomerHistory` može iskoristiti snagu umjetne inteligencije za analizu povijesti transakcija kupca i identificiranje potencijalnih pokazatelja prijevare. To omogućuje sofisticiranije i prilagodljivije tehnike otkrivanja prijevara, jer AI model može učiti i prilagođavati se novim obrascima i anomalijama tijekom vremena.

Ažurirani `FraudDetectionWorker` i klasa `CheckCustomerHistory` pokazuju kako se AI radnici mogu besprijekorno integrirati, poboljšavajući proces otkrivanja prijevara s inteligentnim mogućnostima analize i donošenja odluka.

Analiza Sentimenta Kupaca

Evo još jednog sličnog primjera kako možete implementirati radnika za analizu sentimenta kupaca. Ovaj put s mnogo manje objašnjenja, budući da biste već trebali shvaćati kako ovaj stil programiranja funkcionira:

```
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 primjeru, koraci `CustomerSentimentAnalysisWorker` uključuju predobradu povratnih informacija (npr. uklanjanje šuma, tokenizaciju), provođenje analize sentimenta kako bi se utvrdio općeniti sentiment (pozitivan, negativan ili neutralan), izdvajanje ključnih fraza i tema, identificiranje trendova i obrazaca te generiranje praktičnih uvida temeljenih na analizi.

Primjene u zdravstvu

U području zdravstva, AI radnici mogu pomoći medicinskim stručnjacima i istraživačima u različitim zadacima, što dovodi do poboljšanih ishoda za pacijente i ubrzanih medicinskih otkrića. Neki primjeri uključuju:

Prijem pacijenata

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

Zakazivanje termina: AI radnici mogu upravljati zakazivanjem termina razumijevanjem preferencija pacijenata, dostupnosti i hitnosti njihovih medicinskih potreba. Mogu komunicirati s pacijentima putem konverzacijskih sučelja, vodeći ih kroz proces zakazivanja i pronalazeći najprikladnije termine na temelju zahtjeva pacijenta i dostupnosti zdravstvenog pružatelja usluga.

Prikupljanje medicinske povijesti: Tijekom prijema pacijenata, AI radnici mogu pomoći u prikupljanju i dokumentiranju medicinske povijesti pacijenta. Mogu voditi interaktivne dijaloge s pacijentima, postavljajući relevantna pitanja o njihovim prethodnim medicinskim stanjima, lijekovima, alergijama i obiteljskoj povijesti. AI radnici mogu koristiti tehnike obrade prirodnog jezika za tumačenje i strukturiranje prikupljenih informacija, osiguravajući da su točno zabilježene u elektroničkom zdravstvenom kartonu pacijenta.

Procjena i stratifikacija simptoma: AI radnici mogu provoditi početne procjene simptoma postavljanjem pitanja pacijentima o njihovim trenutnim simptomima, trajanju, ozbiljnosti i povezanim čimbenicima. Koristeći medicinske baze znanja i modele strojnog učenja, ovi radnici mogu analizirati pružene informacije i generirati preliminarne diferencijalne dijagnoze ili preporučiti odgovarajuće sljedeće korake, poput zakazivanja konzultacija sa zdravstvenim djelatnikom ili predlaganja mjera samopomoći.

Provjera osiguranja: AI radnici mogu pomoći pri provjeri osiguranja tijekom prijema pacijenata. Mogu prikupljati podatke o osiguranju pacijenta, komunicirati s osiguravajućim društvima putem API-ja ili web usluga te provjeravati pravo na pokriće i pogodnosti. Ova automatizacija pomaže pojednostaviti proces provjere osiguranja, smanjujući administrativno opterećenje i osiguravajući točno prikupljanje informacija.

Edukacija pacijenata i upute: AI radnici mogu pružiti pacijentima relevantne edukativne materijale i upute na temelju njihovih specifičnih medicinskih stanja ili predstojećih procedura. Mogu isporučivati personalizirani sadržaj, odgovarati na česta pitanja i pružati smjernice o pripremama prije termina, uputama za uzimanje lijekova ili postoperativnoj njezi. To pomaže održati pacijente informiranima i uključenima tijekom njihovog zdravstvenog putovanja.

Korištenjem AI radnika u prijemu pacijenata, zdravstvene organizacije mogu povećati učinkovitost, smanjiti vrijeme čekanja i poboljšati cjelokupno iskustvo pacijenata. Ovi radnici mogu obavljati rutinske zadatke, prikupljati točne informacije i pružati personaliziranu pomoć, omogućujući zdravstvenim djelatnicima da se usredotoče na pružanje visokokvalitetne skrbi pacijentima.

Procjena rizika pacijenata

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

Integracija podataka: AI radnici mogu prikupljati i razumjeti podatke o pacijentima iz više izvora, kao što su elektronički zdravstveni kartoni (EZK), medicinske snimke, laboratorijski rezultati, nosivi uređaji i socijalne odrednice zdravlja. Objedinjavanjem ovih informacija u sveobuhvatni profil pacijenta, AI radnici mogu pružiti holistički pogled na zdravstveno stanje pacijenta i čimbenike rizika.

Stratifikacija rizika: AI radnici mogu koristiti prediktivne modele za stratifikaciju pacijenata u različite kategorije rizika na temelju njihovih individualnih karakteristika i

zdravstvenih podataka. Ova stratifikacija rizika omogućuje zdravstvenim djelatnicima da daju prioritet pacijentima koji zahtijevaju neposredniju pažnju ili intervenciju. Na primjer, pacijenti identificirani kao visokorizični za određeno stanje mogu biti označeni za pojačano praćenje, preventivne mjere ili ranu intervenciju.

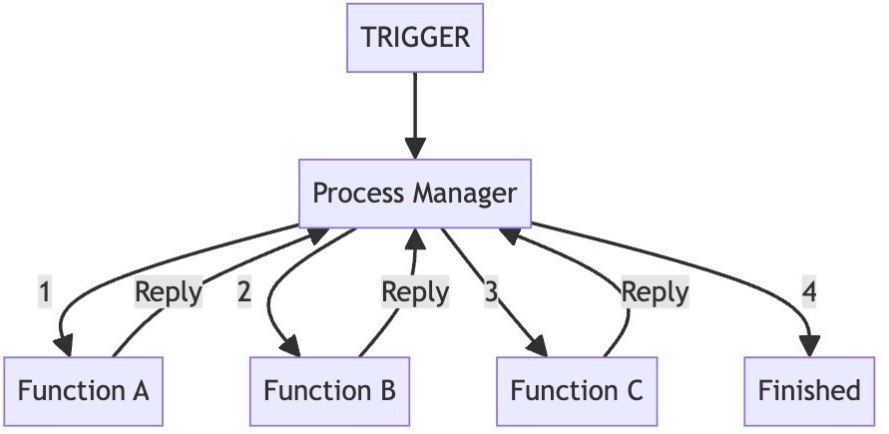
Personalizirani profili rizika: AI radnici mogu generirati personalizirane profile rizika za svakog pacijenta, ističući specifične čimbenike koji doprinose njihovim ocjenama rizika. Ovi profili mogu uključivati uvide u životni stil pacijenta, genetske predispozicije, okolišne čimbenike i socijalne odrednice zdravlja. Pružanjem detaljnog pregleda čimbenika rizika, AI radnici mogu pomoći zdravstvenim djelatnicima da prilagode strategije prevencije i planove liječenja individualnim potrebama pacijenata.

Kontinuirano praćenje rizika: AI radnici mogu kontinuirano pratiti podatke o pacijentima i ažurirati procjene rizika u stvarnom vremenu. Kako nove informacije postaju dostupne, poput promjena u vitalnim znakovima, laboratorijskim rezultatima ili pridržavanju terapije, AI radnici mogu ponovno izračunati ocjene rizika i upozoriti zdravstvene djelatnike na sve značajne promjene. Ovo proaktivno praćenje omogućuje pravovremene intervencije i prilagodbe planova skrbi za pacijente.

Podrška kliničkom odlučivanju: AI radnici mogu integrirati rezultate procjene rizika u sustave podrške kliničkom odlučivanju, pružajući zdravstvenim djelatnicima preporuke i upozorenja temeljene na dokazima. Na primjer, ako ocjena rizika pacijenta za određeno stanje prijeđe određeni prag, AI radnik može potaknuti zdravstvenog djelatnika da razmotri specifične dijagnostičke testove, preventivne mjere ili mogućnosti liječenja na temelju kliničkih smjernica i najboljih praksi.

Ovi radnici mogu obraditi ogromne količine podataka o pacijentima, primijeniti sofisticirane analitičke metode i generirati praktične uvide koji podupiru kliničko odlučivanje. To u konačnici dovodi do poboljšanih ishoda za pacijente, smanjenih troškova zdravstvene skrbi i unaprijeđenog upravljanja zdravljem populacije.

AI radnik kao upravitelj procesa



U kontekstu aplikacija vođenih umjetnom inteligencijom, radnik može biti dizajniran da funkcionira kao Upravitelj procesa, kako je opisano u knjizi "Enterprise Integration Patterns" autora Gregora Hohpea. Upravitelj procesa je središnja komponenta koja održava stanje procesa i određuje sljedeće korake obrade na temelju međurezultata.

Kada AI radnik djeluje kao Upravitelj procesa, prima dolaznu poruku koja inicijalizira proces, poznatu kao *poruka okidač*. AI radnik tada održava stanje izvršenja procesa (kao prijepis razgovora) i upravlja porukom kroz niz koraka obrade implementiranih kao funkcijski alati, koji mogu biti sekvencijalni ili paralelni, i pozivaju se prema njegovoj procjeni.

Ako koristite klasu AI modela poput GPT-4 koja zna kako izvršavati funkcije paralelno, vaš radnik može izvršavati više koraka istovremeno. Doduše, moram priznati da to osobno nisam isprobao i intuicija mi govori da rezultati mogu varirati.

Nakon svakog pojedinačnog koraka obrade, kontrola se vraća natrag AI radniku, omogućujući mu da odredi sljedeće korake obrade na temelju trenutnog stanja i dobivenih rezultata.

Pohranite svoje poruke okidače

Prema mom iskustvu, pametno je implementirati poruku okidač kao objekt podržan bazom podataka. Na taj način svaka instanca procesa je identificirana jedinstvenim primarnim ključem i pruža vam mjesto za pohranu stanja povezanog s izvršavanjem, uključujući AI-jev prijepis razgovora.

Na primjer, evo pojednostavljene verzije Olympijinog modela klase `AccountChange`, koji predstavlja zahtjev za promjenom korisničkog računa.

```
# == 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 obradu zahtjeva za promjenu računa. Primijetite kako se emitira prema Olympijinom Wisper sustavu za objavu/pretplatu nakon što se završi izvršavanje transakcije stvaranja.

Pohrana poruke okidača u bazu podataka na ovaj način pruža trajni zapis svakog zahtjeva za promjenu računa. Svakoj instanci klase `AccountChange` dodjeljuje se jedinstveni primarni ključ, što omogućuje jednostavnu identifikaciju i praćenje pojedinačnih zahtjeva. Ovo je posebno korisno za potrebe revizijskog zapisivanja, jer omogućuje sustavu održavanje povijesnog zapisa svih promjena računa, uključujući kada su zatražene, koje su promjene zatražene i trenutno stanje svakog zahtjeva.

U danom primjeru, klasa `AccountChange` uključuje polja poput `description` za bilježenje detalja zatražene promjene, `state` za predstavljanje trenutnog stanja zahtjeva (npr. zatraženo, završeno, potreban_pregled_čovjeka), i `transcript` za pohranu transkripta AI razgovora vezanog uz zahtjev. Polje `description` je stvarni upit koji se koristi za pokretanje prvog chat dovršetka s AI-jem. Pohrana ovih podataka pruža vrijedan kontekst i omogućuje bolje praćenje i analizu procesa promjene računa.

Pohrana poruka okidača u bazu podataka omogućuje robusno rukovanje pogreškama i oporavak. Ako dođe do pogreške tijekom obrade zahtjeva za promjenu računa, sustav označava zahtjev kao neuspješan i prebacuje ga u stanje koje zahtijeva ljudsku

intervenciju. Time se osigurava da nijedan zahtjev nije izgubljen ili zaboravljen, te da se svi problemi mogu pravilno riješiti.

AI radnik, kao Upravitelj procesa, pruža središnju točku kontrole i omogućuje snažne mogućnosti izvještavanja i otklanjanja grešaka u procesu. Međutim, važno je napomenuti da korištenje AI radnika kao Upravitelja procesa za svaki scenarij tijeka rada u vašoj aplikaciji može biti pretjerano.

Integriranje AI radnika u arhitekturu vaše aplikacije

Pri ugradnji AI radnika u arhitekturu vaše aplikacije, potrebno je riješiti nekoliko tehničkih pitanja kako bi se osigurala glatka integracija i učinkovita komunikacija između AI radnika i drugih komponenti aplikacije. Ovaj odjeljak razmatra ključne aspekte dizajniranja tih sučelja, rukovanja protokom podataka i upravljanja životnim ciklusom AI radnika.

Dizajniranje jasnih sučelja i komunikacijskih protokola

Za olakšavanje besprijekorne integracije između AI radnika i drugih komponenti aplikacije, ključno je definirati jasna sučelja i komunikacijske protokole. Razmotrite sljedeće pristupe:

Integracija temeljena na API-ju: Izložite funkcionalnost AI radnika kroz dobro definirane API-je, poput RESTful krajnjih točaka ili GraphQL shema. Ovo omogućuje drugim komponentama interakciju s AI radnicima korištenjem standardnih HTTP zahtjeva i odgovora. Integracija temeljena na API-ju pruža jasan ugovor između AI radnika i komponenti koje ih koriste, olakšavajući razvoj, testiranje i održavanje integracijskih točaka.

Komunikacija temeljena na porukama: Implementirajte obrasce komunikacije temeljene na porukama, poput redova poruka ili sustava za objavu-pretplatu, kako biste omogućili asinkronu interakciju između AI radnika i drugih komponenti. Ovaj pristup odvaja AI radnike od ostatka aplikacije, omogućujući bolju skalabilnost, otpornost na pogreške i labavo povezivanje. Komunikacija temeljena na porukama posebno je korisna kada je obrada koju izvode AI radnici vremenski zahtjevna ili resursno intenzivna, jer omogućuje drugim dijelovima aplikacije da nastave s izvršavanjem bez čekanja da AI radnici završe svoje zadatke.

Arhitektura vođena događajima: Dizajnirajte svoj sustav oko događaja i okidača koji aktiviraju AI radnike kada se ispune određeni uvjeti. AI radnici mogu se pretplatiti na relevantne događaje i reagirati u skladu s njima, izvršavajući svoje određene zadatke kada se događaji pojave. Arhitektura vođena događajima omogućuje obradu u stvarnom vremenu i omogućuje pozivanje AI radnika prema potrebi, smanjujući nepotrebnu potrošnju resursa. Ovaj pristup je posebno prikladan za scenarije gdje AI radnici trebaju reagirati na specifične akcije ili promjene u stanju aplikacije.

Rukovanje protokom podataka i sinkronizacija

Pri integraciji AI radnika u vašu aplikaciju, ključno je osigurati neometan protok podataka i održavati konzistentnost podataka između AI radnika i drugih komponenti. Razmotrite sljedeće aspekte:

Priprema podataka: Prije unosa podataka u AI radnike, možda ćete trebati izvršiti razne zadatke pripreme podataka, poput čišćenja, formatiranja i/ili transformacije ulaznih podataka. Ne želite samo osigurati da AI radnici mogu učinkovito obrađivati, već i da ne trošite tokene dajući pažnju informacijama koje bi radnik mogao smatrati beskorisnim u najboljem slučaju, ili ometajućim u najgorem. Priprema podataka može uključivati zadatke poput uklanjanja šuma, rukovanja nedostajućim vrijednostima ili pretvaranja tipova podataka.

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

radnika? Razmotrite faktore poput volumena podataka, obrazaca upita i skalabilnosti. Trebate li zadržati transkript AI-ja kao odraz njegovog "procesa razmišljanja" za potrebe revizije ili otklanjanja grešaka, ili je dovoljno imati samo zapis rezultata?

Dohvaćanje podataka: Dobivanje podataka potrebnih radnicima može uključivati upite baza podataka, čitanje iz datoteka ili pristup vanjskim API-jima. Pobrinite se da uzmete u obzir latenciju i način na koji će AI radnici imati pristup najažurnijim podacima. Trebaju li potpuni pristup vašoj bazi podataka ili biste trebali usko definirati opseg njihovog pristupa prema onome što rade? Što je sa skaliranjem? Razmotrite mehanizme predmemoriranja za poboljšanje performansi i smanjenje opterećenja na izvore podataka.

Sinkronizacija podataka: Kada više komponenti, uključujući AI radnike, pristupa i modificira zajedničke podatke, važno je implementirati odgovarajuće mehanizme sinkronizacije kako bi se održala konzistentnost podataka. Strategije zaključavanja baze podataka, poput optimističkog ili pesimističkog zaključavanja, mogu vam pomoći spriječiti konflikte i osigurati integritet podataka. Implementirajte tehnike upravljanja transakcijama za grupiranje povezanih operacija s podacima i održavanje svojstava atomnosti, konzistentnosti, izolacije i trajnosti (ACID).

Upravljanje pogreškama i oporavak: Implementirajte robusne mehanizme za upravljanje pogreškama i oporavak kako biste se nosili s problemima vezanim uz podatke koji se mogu pojaviti tijekom procesa protoka podataka. Elegantno rukujte iznimkama i pružite smislene poruke o pogreškama koje pomažu pri otklanjanju grešaka. Implementirajte mehanizme ponovnih pokušaja i strategije rezervnih rješenja za rukovanje privremenim kvarovima ili prekidima mreže. Definirajte jasne procedure za oporavak i obnovu podataka u slučaju oštećenja ili gubitka podataka.

Pažljivim dizajniranjem i implementacijom mehanizama protoka i sinkronizacije podataka, možete osigurati da vaši AI radnici imaju pristup točnim, konzistentnim i ažurnim podacima. To im omogućuje da učinkovito obavljaju svoje zadatke i proizvode pouzdane rezultate.

Upravljanje životnim ciklusom AI radnika

Razvijte standardizirani proces za inicijalizaciju i konfiguraciju AI radnika. Sklon sam okvirima koji standardiziraju način definiranja postavki kao što su nazivi modela, sistemske direktive i definicije funkcija. Osigurajte da je proces inicijalizacije automatiziran i ponovljiv kako bi se olakšalo implementiranje i skaliranje.

Implementirajte sveobuhvatne mehanizme nadzora i bilježenja za praćenje zdravlja i performansi AI radnika. Prikupljajte metrike poput iskorištenosti resursa, vremena obrade, stope pogrešaka i propusnosti. Koristite centralizirane sustave za bilježenje poput ELK stoga (Elasticsearch, Logstash, Kibana) za agregiranje i analizu zapisa iz više AI radnika.

Ugradite toleranciju na pogreške i otpornost u arhitekturu AI radnika. Implementirajte mehanizme za upravljanje pogreškama i oporavak kako biste elegantno rukovali kvarovima ili iznimkama. Veliki jezični modeli su još uvijek tehnologija u razvoju; pružatelji usluga često prestaju s radom u neočekivanim trenucima. Koristite mehanizme ponovnih pokušaja i prekidače strujnog kruga kako biste spriječili kaskadne kvarove.

Komponibilnost i orkestracija AI radnika

Jedna od ključnih prednosti arhitekture AI radnika je njezina komponibilnost, koja vam omogućuje kombiniranje i orkestriranje više AI radnika za rješavanje složenih problema. Razbijanjem većeg zadatka na manje, upravljivije podzadatke, kojima upravlja specijalizirani AI radnik, možete stvoriti moćne i fleksibilne sustave. U ovom odjeljku istražit ćemo različite pristupe komponiranju i orkestriranju “mnoštva” AI radnika.

Ulančavanje AI radnika za višekoračne tijekove rada

U mnogim scenarijima, složeni zadatak može se rastaviti na niz sekvencijalnih koraka, gdje izlaz jednog AI radnika postaje ulaz za sljedećeg. Ovo ulančavanje AI radnika stvara

višekoračni tijek rada ili cjevovod. Svaki AI radnik u lancu fokusira se na specifični podzadatak, a konačni izlaz je rezultat kombiniranih napora svih radnika.

Razmotrimo primjer u kontekstu Ruby on Rails aplikacije za obradu korisnički generiranog sadržaja. Tijek rada uključuje sljedeće korake, koji su, priznajemo, vjerojatno pojedinačno previše jednostavni da bi ih vrijedilo raščlanjivati na ovaj način u stvarnim slučajevima korištenja, ali čine primjer lakšim za razumijevanje:

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

2. **Otkrivanje jezika**: AI radnik koji identificira jezik očišćenog teksta.

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

4. **Kategorizacija sadržaja**: AI radnik koji klasificira tekst u unaprijed definirane kategorije koristeći tehnike obrade prirodnog jezika.

Evo vrlo pojednostavljenog primjera kako možete ulančati ove AI radnike koristeći 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 primjeru, klasa `ContentProcessor` inicijalizira se sa sirovim tekstom i povezuje AI radnike zajedno u metodi `process`. Svaki AI radnik izvršava svoj specifični

zadatak i prosljeđuje rezultat sljedećem radniku u lancu. Konačni izlaz je hash koji sadrži očišćeni tekst, detektirani jezik, sentiment i kategoriju sadržaja.

Paralelna obrada za nezavisne AI radnike

U prethodnom primjeru, AI radnici su povezani sekvencijalno, gdje svaki radnik obrađuje tekst i prosljeđuje rezultat sljedećem radniku. Međutim, ako imate više AI radnika koji mogu raditi nezavisno na istom ulazu, možete optimizirati tijek rada obrađujući ih paralelno.

U danom scenariju, nakon što `TextCleanupWorker` izvrši čišćenje teksta, `LanguageDetectionWorker`, `SentimentAnalysisWorker` i `CategorizationWorker` mogu svi neovisno obrađivati očišćeni tekst. Pokretanjem ovih radnika paralelno, možete potencijalno smanjiti ukupno vrijeme obrade i poboljšati učinkovitost vašeg tijeka rada.

Za postizanje paralelne obrade u Rubyju, možete iskoristiti tehnike konkurentnosti kao što su dretve ili asinkrono programiranje. Evo primjera kako možete modificirati klasu `ContentProcessor` da obrađuje završna tri radnika paralelno koristeći dretve:

```
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 optimiziranoj verziji koristimo biblioteku `concurrent-ruby` za stvaranje `Concurrent::Future` objekata za svakog od neovisnih AI radnika. „Future“ predstavlja izračun koji će se izvršiti asinkrono u zasebnoj dretvi.

Nakon koraka čišćenja teksta, stvaramo tri `Future` objekta: `language_future`, `sentiment_future` i `category_future`. Svaki `Future` izvršava svog odgovarajućeg AI radnika (`LanguageDetectionWorker`, `SentimentAnalysisWorker` i `CategorizationWorker`) u zasebnoj dretvi, prosljeđujući `cleaned_text` kao ulaz.

Pozivanjem metode `value` na svakom `Future` objektu, čekamo da se izračun dovrši i dohvaćamo rezultat. Metoda `value` blokira izvršavanje dok rezultat nije dostupan, osiguravajući da su svi paralelni radnici završili obradu prije nastavka.

Na kraju, konstruiramo izlazni hash s očišćenim tekstom i rezultatima paralelnih radnika, baš kao i u izvornom primjeru.

Obradom neovisnih AI radnika paralelno, možete potencijalno smanjiti ukupno vrijeme obrade u usporedbi sa sekvencijalnim izvršavanjem. Ova optimizacija posebno je korisna kada se radi o vremenski zahtjevnim zadacima ili pri obradi velikih količina podataka.

Međutim, važno je napomenuti da stvarni dobici u performansama ovise o različitim

čimbenicima, kao što su složenost svakog radnika, dostupni sistemski resursi i režijski troškovi upravljanja dretvama. Uvijek je dobra praksa provesti mjerenje performansi i profiliranje koda kako biste odredili optimalnu razinu paralelizma za vaš specifični slučaj korištenja.

Dodatno, pri implementaciji paralelne obrade, vodite računa o svim dijeljenim resursima ili međuovisnostima između radnika. Osigurajte da radnici mogu djelovati neovisno bez sukoba ili uvjeta utrke. Ako postoje međuovisnosti ili dijeljeni resursi, možda ćete trebati implementirati odgovarajuće mehanizme sinkronizacije kako biste održali integritet podataka i izbjegli probleme poput zastoja ili nedosljednih rezultata.

Rubyjeva Globalna brava interpretera i asinkrona obrada

Važno je razumjeti implikacije Rubyjeve Globalne brave interpretera (GIL) kada razmatramo asinkronu obradu temeljenu na dretvama u Rubyju.

GIL je mehanizam u Rubyjevom interpreteru koji osigurava da samo jedna dretva može izvršavati Ruby kod u određenom trenutku, čak i na procesorima s više jezgri. To znači da iako se unutar Ruby procesa može stvoriti i upravljati više dretvi, samo jedna dretva može aktivno izvršavati Ruby kod u bilo kojem trenutku.

GIL je dizajniran da pojednostavi implementaciju Ruby interpretera i pruži sigurnost dretvi za Rubyjeve interne strukture podataka. Međutim, također ograničava mogućnost stvarnog paralelnog izvršavanja Ruby koda.

Kada koristite dretve u Rubyju, kao što je to s bibliotekom `concurrent-ruby` ili ugrađenom klasom `Thread`, dretve su podložne ograničenjima GIL-a. GIL omogućuje svakoj dretvi da izvršava Ruby kod tijekom kratkog vremenskog odsječka prije prebacivanja na drugu dretvu, stvarajući iluziju istodobnog izvršavanja.

Međutim, zbog GIL-a, stvarno izvršavanje Ruby koda ostaje sekvencijalno. Dok jedna

dretva izvršava Ruby kod, druge dretve su u biti pauzirane, čekajući svoj red za dobivanje GIL-a i izvršavanje.

To znači da je asinkrona obrada temeljena na dretvama u Rubyju najučinkovitija za zadatke vezane uz U/I, kao što su čekanje na odgovore vanjskih API-ja (poput jezičnih modela velikih razmjera koje hostaju treće strane) ili izvođenje U/I operacija s datotekama. Kada dretva naiđe na U/I operaciju, može osloboditi GIL, omogućujući drugim dretvama da se izvršavaju dok čekaju završetak U/I operacije.

S druge strane, za zadatke vezane uz procesor, kao što su intenzivni izračuni ili dugotrajna obrada AI radnika, GIL može ograničiti potencijalne dobitke u performansama kod paralelizma temeljenog na dretvama. Budući da samo jedna dretva može izvršavati Ruby kod u određenom trenutku, ukupno vrijeme izvršavanja možda neće biti značajno smanjeno u usporedbi sa sekvencijalnom obradom.

Za postizanje stvarnog paralelnog izvršavanja za zadatke vezane uz procesor u Rubyju, možda ćete trebati istražiti alternativne pristupe, kao što su:

- Korištenje paralelizma temeljenog na procesima s više Ruby procesa, od kojih svaki radi na zasebnoj jezgri procesora.
- Iskorištavanje vanjskih biblioteka ili okvira koji pružaju nativna proširenja ili sučelja prema jezicima bez GIL-a, kao što su C ili Rust.,
- Korištenje okvira za distribuirano računanje ili redova poruka za distribuciju zadataka preko više strojeva ili procesa.

Ključno je razmotriti prirodu vaših zadataka i ograničenja koja nameće GIL pri dizajniranju i implementaciji asinkrone obrade u Rubyju. Iako asinkrona obrada temeljena na dretvama može pružiti prednosti za zadatke vezane uz U/I, možda neće ponuditi značajna poboljšanja performansi za zadatke vezane uz procesor zbog ograničenja GIL-a.

Tehnike ansambla za poboljšanu točnost

Tehnike ansambla uključuju kombiniranje izlaza više AI radnika kako bi se poboljšala ukupna točnost ili robusnost sustava. Umjesto oslanjanja na jednog AI radnika, tehnike ansambla koriste kolektivnu inteligenciju više radnika za donošenje informiranijih odluka.

Ansambli su posebno važni ako različiti dijelovi vašeg radnog toka najbolje funkcioniraju s različitim AI modelima, što je češća pojava nego što biste mogli pomisliti. Moćni modeli poput GPT-4 izuzetno su skupi u usporedbi s manje sposobnim opcijama otvorenog koda i vjerojatno nisu potrebni za svaki pojedini korak radnog toka vaše aplikacije.

Jedna uobičajena tehnika ansambla je većinsko glasovanje, gdje više AI radnika neovisno obrađuje isti unos, a konačni rezultat određuje se većinskim konsenzusom. Ovaj pristup može pomoći u ublažavanju utjecaja pojedinačnih pogrešaka radnika i poboljšati ukupnu pouzdanost sustava.

Razmotrimo primjer gdje imamo tri AI radnika za analizu sentimenta, od kojih svaki koristi različiti model ili je opremljen različitim kontekstima. Možemo kombinirati njihove rezultate koristeći većinsko glasovanje 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 primjeru, klasa `SentimentAnalysisEnsemble`. inicijalizira se s 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 dobiva najviše glasova od ansambla radnika.

Ansambli su očito slučaj gdje eksperimentiranje s paralelizmom može biti vrijedno vašeg vremena.

Dinamički odabir i pozivanje AI radnika

U nekim, ako ne i većini slučajeva, odabir specifičnog AI radnika može ovisiti o uvjetima tijekom izvođenja ili korisničkim unosima. Dinamički odabir i pozivanje AI radnika omogućuju fleksibilnost i prilagodljivost sustava.

Možda ćete biti u iskušenju da pokušate ugraditi mnogo funkcionalnosti u jednog AI radnika, dajući mu mnogo funkcija i veliki kompliciran prompt koji objašnjava kako ih pozvati. Oduprite se iskušenju, vjerujte mi. Jedan od razloga zašto se pristup o kojem raspravljamo u ovom poglavlju zove "Mnoštvo radnika" jest da nas podsjeti kako je poželjno imati mnoštvo specijaliziranih radnika, od kojih svaki obavlja svoj mali posao u službi veće svrhe.

Na primjer, razmotrite chatbot aplikaciju gdje su različiti AI radnici odgovorni za obradu različitih vrsta korisničkih upita. Na temelju korisničkog unosa, aplikacija dinamički odabire 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 primjeru, `ChatbotController` prima korisnički upit kroz akciju `process_query`. Prvo koristi `QueryClassifierWorker` za određivanje vrste upita. Na temelju klasificirane vrste upita, kontroler dinamički odabire odgovarajućeg AI radnika za generiranje odgovora. Ovaj dinamički odabir omogućuje chatbotu da obrađuje različite vrste upita i usmjerava ih prema relevantnim AI radnicima.

Budući da je rad `QueryClassifierWorker`-a relativno jednostavan i ne zahtijeva puno konteksta ili definicija funkcija, vjerojatno ga možete implementirati koristeći ultra-brzi mali LLM poput `mistralai/mixtral-8x7b-instruct:nitro`. Njegove mogućnosti su blizu razini GPT-4 za mnoge zadatke i, u trenutku pisanja ovog teksta, Groq ga može posluživati nevjerojatnom propusnošću od 444 tokena u sekundi.

Kombiniranje tradicionalnog NLP-a s LLM-ovima

Iako su Veliki jezični modeli (LLM) revolucionirali područje obrade prirodnog jezika (NLP), nudeći neusporedivu svestranost i performanse u širokom rasponu zadataka, oni nisu uvijek najučinkovitije ili troškovno najisplativije rješenje za svaki problem. U mnogim slučajevima, kombiniranje tradicionalnih NLP tehnika s LLM-ovima može dovesti do optimiziranijih, ciljanijih i ekonomičnijih pristupa rješavanju specifičnih NLP izazova.

Zamislite LLM-ove kao švicarske nožiće NLP-a—nevjerojatno svestrane i moćne, ali ne nužno najbolji alat za svaki posao. Ponekad, namjenski alat poput vadičepa ili otvarača za konzerve može biti učinkovitiji za određeni zadatak. Slično tome, tradicionalne NLP tehnike, poput grupiranja dokumenata, identifikacije tema i klasifikacije, često mogu pružiti ciljanija i troškovno učinkovitija rješenja za određene aspekte vašeg NLP postupka.

Jedna od ključnih prednosti tradicionalnih NLP tehnika je njihova računalna učinkovitost. Ove metode, koje se često oslanjaju na jednostavnije statističke modele ili pristupe temeljene na pravilima, mogu obrađivati velike količine tekstualnih podataka mnogo brže i s manjim računalnim opterećenjem u usporedbi s LLM-ovima. To ih čini posebno pogodnima za zadatke koji uključuju analizu i organizaciju velikih korpusa dokumenata, poput grupiranja sličnih članaka ili identificiranja ključnih tema unutar zbirke tekstova.

Štoviše, tradicionalne NLP tehnike često mogu postići visoku točnost i preciznost za specifične zadatke, posebno kada su trenirane na domenski specifičnim skupovima podataka. Na primjer, dobro podešeni klasifikator dokumenata koji koristi tradicionalne algoritme strojnog učenja poput Metode potpornih vektora (SVM) ili Naivnog Bayesa može precizno kategorizirati dokumente u predefinirane kategorije uz minimalan računalni trošak.

Međutim, LLM-ovi stvarno zablistaju kada se radi o zadacima koji zahtijevaju dublje razumijevanje jezika, konteksta i zaključivanja. Njihova sposobnost generiranja koherentnog i kontekstualno relevantnog teksta, odgovaranja na pitanja i sažimanja dugih odlomaka je nenadmašna tradicionalnim NLP metodama. LLM-ovi mogu učinkovito upravljati složenim jezičnim pojavama, poput dvosmislenosti, koreferencije i idiomatskih izraza, čineći ih neprocjenjivima za zadatke koji zahtijevaju generiranje ili razumijevanje prirodnog jezika.

Prava snaga leži u kombiniranju tradicionalnih NLP tehnika s LLM-ovima kako bi se stvorili hibridni pristupi koji iskorištavaju prednosti obaju. Korištenjem tradicionalnih NLP metoda za zadatke poput predobrade dokumenata, grupiranja i ekstrakcije tema, možete učinkovito organizirati i strukturirati svoje tekstualne podatke. Te strukturirane informacije zatim se mogu proslijediti LLM-ovima za naprednije zadatke, poput generiranja sažetaka, odgovaranja na pitanja ili stvaranja sveobuhvatnih izvještaja.

Na primjer, razmotrimo slučaj uporabe gdje želite generirati izvještaj o trendovima za određenu domenu na temelju velikog korpusa pojedinačnih dokumenata o trendovima. Umjesto da se oslanjate isključivo na LLM-ove, što može biti računalno skupo i dugotrajno za obradu velikih količina teksta, možete primijeniti hibridni pristup:

1. Koristite tradicionalne NLP tehnike, poput modeliranja tema (npr. Latentna Dirichletova alokacija) ili algoritama grupiranja (npr. K-means), za grupiranje sličnih dokumenata o trendovima i identificiranje ključnih tema unutar korpusa.
2. Proslijedite grupirane dokumente i identificirane teme LLM-u, koristeći njegove superiorne sposobnosti razumijevanja i generiranja jezika za stvaranje

koherentnih i informativnih sažetaka za svaku grupu ili temu.

3. Na kraju, koristite LLM za generiranje sveobuhvatnog izvještaja o trendovima kombiniranjem pojedinačnih sažetaka, ističući najznačajnije trendove i pružajući uvide i preporuke na temelju agregiranih informacija.

Kombiniranjem tradicionalnih NLP tehnika s LLM-ovima na ovaj način, možete učinkovito obrađivati velike količine tekstualnih podataka, izvlačiti smislene uvide i generirati visokokvalitetne izvještaje uz optimizaciju računalnih resursa i troškova.

Dok se upuštate u svoje NLP projekte, ključno je pažljivo procijeniti specifične zahtjeve i ograničenja svakog zadatka te razmotriti kako se tradicionalne NLP metode i LLM-ovi mogu zajedno iskoristiti za postizanje najboljih rezultata. Kombiniranjem učinkovitosti i preciznosti tradicionalnih tehnika s versatilnošću i snagom LLM-ova, možete stvoriti visoko učinkovita i ekonomična NLP rješenja koja donose vrijednost vašim korisnicima i dionicima.

Korištenje alata

U području razvoja aplikacija temeljenih na umjetnoj inteligenciji, koncept “korištenja alata” ili “pozivanja funkcija” pojavio se kao moćna tehnika koja omogućuje vašem LLM-u povezivanje s vanjskim alatima, API-jima, funkcijama, bazama podataka i drugim resursima. Ovaj pristup omogućuje bogatiji skup ponašanja od samog generiranja teksta te dinamičnije interakcije između vaših AI komponenti i ostatka ekosustava vaše aplikacije. Kao što ćemo razmotriti u ovom poglavlju, korištenje alata također vam pruža mogućnost da vaš AI model generira podatke na strukturirane načine.

Što je korištenje alata?

Korištenje alata, poznato i kao pozivanje funkcija, tehnika je koja omogućuje programerima definiranje popisa funkcija s kojima LLM može komunicirati tijekom

procesa generiranja. Ovi alati mogu varirati od jednostavnih pomoćnih funkcija do složenih API-ja ili upita baza podataka. Pružanjem LLM-u pristupa ovim alatima, programeri mogu proširiti mogućnosti modela i omogućiti mu izvršavanje zadataka koji zahtijevaju vanjsko znanje ili akcije.

Slika 8. Primjer 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 uporabe alata jest dati VJM-u mogućnost dinamičkog odabira i izvršavanja odgovarajućih alata na temelju korisničkog unosa ili zadatka koji treba obaviti. Umjesto da se oslanja isključivo na prethodno uvježbano znanje modela, uporaba alata omogućuje VJM-u korištenje vanjskih resursa za generiranje točnijih,

relevantnijih i praktičnijih odgovora. Uporaba alata čini tehnike poput dohvatom potpomognutog generiranja (RAG) mnogo lakšima za implementaciju nego što bi bile inače.

> Imajte na umu da, osim ako nije drugačije navedeno, ova knjiga pretpostavlja da vaš AI model nema pristup ugrađenim alatima na strani poslužitelja. Sve alate koje želite učiniti dostupnima vašem AI-ju morate eksplicitno deklarirati u svakom API zahtjevu, s odredbama za njihovo izvršavanje ako i kada vam AI kaže da bi želio koristiti taj alat u svom odgovoru.

Potencijal uporabe alata

Uporaba alata otvara širok spektar mogućnosti za aplikacije temeljene na umjetnoj inteligenciji. Evo nekoliko primjera što se može postići uporabom alata:

1. **Chatbotovi i virtualni asistenti**: Povezivanjem VJM-a s vanjskim alatima, chatbotovi i virtualni asistenti mogu izvršavati složenije zadatke, poput dohvaćanja informacija iz baza podataka, izvršavanja API poziva ili interakcije s drugim sustavima. Na primjer, chatbot može koristiti CRM alat za promjenu statusa ponude na temelju korisničkog zahtjeva.
2. **Analiza podataka i uvidi**: VJM-ovi se mogu povezati s alatima ili bibliotekama za analizu podataka kako bi izvršavali napredne zadatke obrade podataka. To omogućuje aplikacijama generiranje uvida, provođenje komparativnih analiza ili pružanje preporuka temeljenih na podacima na temelju korisničkih upita.
3. **Pretraživanje i dohvat informacija**: Uporaba alata omogućuje VJM-ovima interakciju s tražilicama, vektorskim bazama podataka ili drugim sustavima za dohvat informacija. Pretvaranjem korisničkih upita u upite za pretraživanje,

VJM može dohvatiti relevantne informacije iz više izvora i pružiti sveobuhvatne odgovore na korisnička pitanja.

4. **Integracija s vanjskim uslugama**: Uporaba alata omogućuje besprijekornu integraciju između aplikacija temeljenih na umjetnoj inteligenciji i vanjskih usluga ili API-ja. Na primjer, VJM bi mogao komunicirati s API-jem za vremenske prognoze kako bi pružio ažuriranja o vremenu u stvarnom vremenu ili s API-jem za prevođenje kako bi generirao višejezične odgovore.

Tijek rada uporabe alata

Tijek rada uporabe alata obično uključuje četiri ključna koraka:

1. Uključivanje definicija funkcija u kontekst zahtjeva
2. Dinamički (ili eksplicitni) odabir alata
3. Izvršavanje funkcija
4. Opcionalni nastavak izvornog upita

Pregledajmo detaljno svaki od ovih koraka.

Uključivanje definicija funkcija u kontekst zahtjeva

AI zna koje alate ima na raspolaganju jer mu dajete popis kao dio vašeg zahtjeva za dovršavanje (obično definirano kao funkcije koristeći varijantu JSON sheme).

Precizna sintaksa definicije alata specifična je za model.

Ovako se definira 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 evo kako biste definirali istu funkciju za GPT-4, prosljeđujući je kao vrijednost 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"],
```

```
    },
}
```

> Gotovo isto, samo drugačije bez očitog razloga! Kako iritantno.

Definicije funkcija određuju naziv, opis i ulazne parametre. Ulazni parametri mogu se dodatno definirati pomoću atributa kao što su enumeracije za ograničavanje prihvatljivih vrijednosti te određivanjem je li parametar obavezan ili ne.

Uz same definicije funkcija, možete također uključiti upute ili kontekst o tome zašto i kako koristiti funkciju u sistemskoj direktivi.

Na primjer, moj alat za pretraživanje weba u Olympiji uključuje ovu sistemsku direktivu, koja podsjeća UI da ima spomenute 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 čimbenikom u performansama alata. Vaši opisi trebali bi objasniti svaki detalj o alatu, uključujući:

- Što alat radi
- Kada ga treba koristiti (a kada ne)
- Što znači svaki parametar i kako utječe na ponašanje alata
- Sva važna upozorenja ili ograničenja koja se odnose na implementaciju alata

Što više konteksta možete dati umjetnoj inteligenciji o svojim alatima, to će ona biti bolja u odlučivanju kada i kako ih koristiti. Primjerice, Anthropic preporučuje najmanje 3-4 rečenice po opisu alata za svoj Claude 3 seriju, a i više ako je alat složen.

Nije nužno intuitivno, ali opisi se također smatraju važnijima od primjera. Iako možete uključiti primjere kako koristiti alat u njegovom opisu ili u pratećem upitu, to je manje važno od jasnog i sveobuhvatnog objašnjenja svrhe i parametara alata. Primjere dodajte tek nakon što ste u potpunosti razradili opis.

Evo primjera specifikacije API funkcije nalik na Stripe:

```
{
  "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 s ugniježđenim specifikacijama funkcija i složenim izlaznim tipovima podataka kao što su polja, rječnici itd. No teoretski, trebali biste moći dostaviti JSON Schema specifikacije proizvoljne dubine!

Dinamički odabir alata

Kada izvršavate chat completion koji uključuje definicije alata, LLM dinamički odabire najprikladniji alat(e) za korištenje i generira potrebne ulazne parametre za svaki alat.

U praksi, sposobnost AI-ja da pozove *točno* pravu funkciju i *točno* slijedi vašu specifikaciju za ulazne podatke je promjenjiva. Postavljanje temperature hiperparametra skroz na 0.0 značajno pomaže, ali prema mom iskustvu i dalje ćete povremeno dobivati pogreške. Te pogreške uključuju halucinirana imena funkcija, pogrešno imenovane ili jednostavno nedostajuće ulazne parametre. Parametri se prenose kao JSON, što znači da ćete ponekad vidjeti pogreške uzrokovane skraćenim, pogrešno navedenim ili na drugi način neispravnim JSON-om.

Obrasci Samooporavljajućih podataka mogu pomoći pri automatskom popravljanju poziva funkcija koji se prekidaju zbog sintaksnih pogrešaka.

Prisilni (tzv. Eksplicitni) odabir alata

Neki modeli vam daju mogućnost prisilnog pozivanja određene funkcije kao parametar u zahtjevu. U suprotnom, hoće li se funkcija pozvati ili ne, u potpunosti ovisi o prosudbi AI-ja.

Mogućnost prisilnog pozivanja funkcije ključna je u određenim scenarijima gdje želite osigurati 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 unaprijed definiranom tijeku rada koji zahtijeva izvršavanje određene funkcije u određeno vrijeme. Prisilnim pozivanjem možete jamčiti da će željena funkcija biti pozvana umjesto da ljubazno molite AI da to učini.
2. **Debugiranje i testiranje**: Prilikom razvoja i testiranja aplikacija temeljenih na AI-ju, mogućnost prisilnog pozivanja funkcija je neprocjenjiva za potrebe debugiranja. Eksplicitnim pokretanjem određenih funkcija možete izolirati i testirati pojedine komponente vaše aplikacije. To vam omogućuje provjeru ispravnosti implementacija funkcija, validaciju ulaznih parametara i osiguravanje da se vraćaju očekivani rezultati.
3. **Rukovanje rubnim slučajevima**: Mogu postojati rubni slučajevi ili iznimne situacije gdje proces dinamičkog odabira AI-ja možda neće odabrati izvršavanje funkcije koju bi trebao, a vi to znate na temelju vanjskih procesa. U takvim slučajevima, mogućnost prisilnog pozivanja funkcije omogućuje vam eksplicitno rukovanje tim situacijama. Definirajte pravila ili uvjete u logici vaše aplikacije kako biste odredili kada pregaziti prosudbu AI-ja.

4. **Konzistentnost i reproducibilnost**: Ako imate određeni slijed funkcija koje se trebaju izvršiti određenim redoslijedom, prisilno pozivanje jamči da će se isti slijed pratiti svaki put. Ovo je posebno važno u aplikacijama gdje su konzistentnost i predvidljivo ponašanje kritični, kao što su financijski sustavi ili znanstvene simulacije.
5. **Optimizacija performansi**: U nekim slučajevima, prisilno 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 nepotrebno opterećenje, možete zaobići proces odabira i izravno pozvati potrebnu funkciju. To može pomoći u smanjenju latencije i poboljšanju ukupne učinkovitosti vaše aplikacije.

Ukratko, mogućnost prisilnog pozivanja funkcija u aplikacijama temeljenim na AI-ju pruža eksplicitnu kontrolu, pomaže u debugiranju i testiranju, rukuje rubnim slučajevima, osigurava konzistentnost i reproducibilnost. To je moćan alat u vašem arsenalu, ali moramo raspraviti još jedan aspekt ove važne značajke.

U mnogim slučajevima donošenja odluka, uvijek želimo da model napravi poziv funkcije i možda nikada ne želimo da model odgovori samo svojim internim znanjem. Na primjer, ako preusmjeravate između više modela specijaliziranih za različite zadatke (višejezični unos, matematika, itd.), možete koristiti model koji poziva funkcije za delegiranje zahtjeva jednom od pomoćnih modela i nikada ne odgovara samostalno.

Parametar tool_choice

GPT-4 i drugi jezični modeli koji podržavaju pozivanje funkcija daju vam parametar `tool_choice` za kontrolu je li upotreba alata obavezna kao dio završetka. Ovaj parametar ima tri moguće vrijednosti:

- `auto` daje AI-ju punu diskreciju oko korištenja alata ili jednostavnog odgovaranja
- `required` govori AI-ju da *mora* pozvati alat *umjesto* odgovaranja, ali ostavlja odabir alata AI-ju
- Treća opcija je postaviti parametar `name_of_function` koji želite prisilno pozvati. Više o tome u sljedećem odjeljku.

Imajte na umu da ako postavite izbor alata na `required`, model će biti prisiljen odabrati najrelevantniju funkciju za poziv među onima koje su mu dostupne, čak i ako nijedna zapravo ne odgovara upitu. U vrijeme objave, nisam upoznat s modelom koji bi vratio prazan `tool_calls` odgovor ili na neki drugi način dao do znanja da nije pronašao odgovarajuću funkciju za poziv.

Prisilno Korištenje Funkcije za Dobivanje Strukturiranog Izlaza

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

> Zašto je prisilno korištenje funkcija za dobivanje strukturiranog izlaza toliko važno? Jednostavno rečeno, zato što je izvlačenje strukturiranih podataka iz LLM izlaza vrlo naporno. Možete si malo olakšati život tražeći podatke u XML formatu, ali tada morate parsirati XML. I što učiniti kada taj XML nedostaje jer je vaša UI odgovorila: "Žao mi je, ali ne mogu generirati zatražene podatke jer bla, bla, bla..."

Kada na ovaj način koristite alate:

- Vjerojatno biste trebali definirati samo jedan alat u svom zahtjevu
- Ne zaboravite prisiliti korištenje njegove funkcije pomoću parametra `tool_choice`
- Zapamtite da će model proslijediti unos alatu, pa naziv alata i opis trebaju biti iz perspektive modela, ne vaše

Ova posljednja točka zaslužuje primjer radi jasnoće. Recimo da tražite UI da napravi analizu sentimenta korisničkog teksta. Naziv funkcije ne bi bio `analyze_sentiment`, već nešto poput `save_sentiment_analysis`. UI je taj koji radi analizu sentimenta, *ne alat*. Sve što alat radi (iz perspektive UI-ja) jest spremanje rezultata analize.

Evo primjera korištenja Claude 3 za bilježenje sažetka slike u dobro strukturirani JSON, ovaj put iz naredbenog retka 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 prikazanom primjeru koristimo Claude 3 model od Anthropica za generiranje strukturiranog JSON sažetka slike. Evo kako to funkcionira:

1. Definiramo jedan alat nazvan `record_summary` u polju `tools` zahtjeva. Ovaj alat je zadužen za bilježenje sažetka slike u dobro strukturiranom JSON formatu.
2. Alat `record_summary` ima `input_schema` koji određuje očekivanu strukturu JSON izlaza. Definira tri svojstva:
    - `key_colors`: Polje objekata koji predstavljaju ključne boje u slici. Svaki objekt boje ima svojstva za crvenu, zelenu i plavu vrijednost (u rasponu 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`: Neobavezno svojstvo tipa integer za procijenjenu godinu kada je slika snimljena, ako se čini da je riječ o nefikcijskoj fotografiji.
3. U polju `messages` dostavljamo podatke slike kao base64-kodirani string zajedno s tipom medija. Ovo omogućuje modelu da obradi sliku kao dio ulaza.
4. Također potičemo Claudea da koristi alat `record_summary` za opisivanje slike.
5. Kada se zahtjev pošalje Claude 3 modelu, on analizira sliku i generira JSON sažetak temeljen na specificiranom `input_schema`. Model izdvaja ključne boje, pruža kratak opis i procjenjuje godinu kada je slika snimljena (ako je primjenjivo).

6. Generirani JSON sažetak prenosi se kao parametri alatu `record_summary`, pružajući strukturirani prikaz ključnih karakteristika slike.

Korištenjem alata `record_summary` s dobro definiranim `input_schema`, možemo dobiti strukturirani JSON sažetak slike bez oslanjanja na ekstrakciju običnog teksta. Ovaj pristup osigurava da izlaz slijedi konzistentan format i može se lako parsirati i obraditi u nizvodnim komponentama aplikacije.

Mogućnost prisiljavanja poziva funkcije i određivanja očekivane izlazne strukture moćna je značajka korištenja alata u aplikacijama vođenim umjetnom inteligencijom. Omogućuje programerima da imaju više kontrole nad generiranim izlazom i pojednostavljuje integraciju podataka generiranih umjetnom inteligencijom u tijek rada njihove aplikacije.

Izvršavanje funkcije(a)

Definirali ste funkcije i dali upit vašem UI-ju, koji je odlučio da treba pozvati jednu od vaših funkcija. Sada je vrijeme da vaš aplikacijski kod ili biblioteka, ako koristite Ruby gem poput `raix-rails`, proslijedi poziv funkcije i njezine parametre odgovarajućoj implementaciji *u vašem aplikacijskom kodu.*

Vaš aplikacijski kod odlučuje što učiniti s rezultatima izvršavanja funkcije. Možda to uključuje jednu liniju koda u lambda izrazu, ili možda uključuje pozivanje vanjskog API-ja. Možda uključuje pozivanje druge UI komponente, ili možda uključuje stotine ili čak tisuće linija koda u ostatku vašeg sustava. To je u potpunosti na vama.

Ponekad je poziv funkcije kraj operacije, ali ako rezultati predstavljaju informacije u lancu razmišljanja koje UI treba nastaviti, tada vaš aplikacijski kod treba umetnuti rezultate izvršavanja u transkript razgovora i pustiti UI da nastavi s obradom.

Na primjer, evo Raix deklaracije funkcije koju koristi Olympijin `AccountManager` za komunikaciju s našim klijentima kao dio Inteligentne orkestracije radnog tijeka 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 što se ovdje događa, pa ću to razložiti.

1. Klasa `AccountManager` definira mnoge funkcije vezane uz upravljanje računom. Može promijeniti vaš plan, dodavati i uklanjati članove tima, među ostalim stvarima.
2. Njegove upute najviše razine govore `AccountManager`-u da treba obavijestiti vlasnika računa o rezultatima zahtjeva za promjenom računa, koristeći funkciju `notify_account_owner`.
3. Sažeta definicija funkcije uključuje njezin:

- naziv
- opis
- parametre `message: { type: "string" }`
- blok koji se izvršava kada se funkcija pozove

Nakon ažuriranja transkripta s rezultatima funkcijskog bloka, ponovno se poziva metoda `chat_completion`. Ova metoda je odgovorna za slanje ažuriranog transkripta razgovora natrag AI modelu za daljnju obradu. Ovaj proces nazivamo *konverzacijskom petljom*.

Kada AI model primi novi zahtjev za dovršavanje razgovora s ažuriranim transkriptom, ima pristup rezultatima prethodno izvršene funkcije. Može analizirati te rezultate, uključiti ih u svoj proces odlučivanja i generirati sljedeći odgovor ili radnju na temelju kumulativnog konteksta razgovora. Može odabrati izvršavanje dodatnih funkcija na temelju ažuriranog konteksta, ili može generirati konačni odgovor na izvorni upit ako utvrdi da dodatni pozivi funkcija nisu potrebni.

Neobavezni nastavak izvornog upita

Kada pošaljete rezultate alata natrag VJM-u i nastavite s obradom izvornog upita, AI koristi te rezultate za pozivanje dodatnih funkcija ili generiranje konačnog tekstualnog odgovora.

Neki modeli poput Cohereovog Command-R mogu citirati specifične alate koje su koristili u svojim odgovorima, pružajući dodatnu transparentnost i sljedivost.

Ovisno o modelu koji se koristi, rezultati poziva funkcije će se nalaziti u porukama transkripta koje imaju svoju posebnu ulogu ili će biti prikazani u nekoj drugoj sintaksi. Ali važno je da ti podaci budu u transkriptu, kako bi ih AI mogao uzeti u obzir pri odlučivanju što dalje učiniti.

Česta (i potencijalno skupa) pogreška je zaboraviti dodati rezultate funkcije u transkript prije nastavka razgovora. Kao rezultat toga, AI će biti potaknut na gotovo isti način kao i prije nego što je prvi put pozvao funkciju. Drugim riječima, što se tiče AI-ja, još nije pozvao funkciju. Pa je pozove ponovno. I ponovno. I ponovno, zauvijek dok ga ne prekinete. Nadajmo se da vaš kontekst nije bio prevelik, a vaš model nije bio preskup!

Najbolje prakse za korištenje alata

Da biste izvukli najviše iz korištenja alata, razmotrite sljedeće najbolje prakse.

Opisne definicije

Osigurajte jasne i opisne nazive i opise za svaki alat i njegove ulazne parametre. To pomaže VJM-u da bolje razumije svrhu i mogućnosti svakog alata.

> Iz iskustva vam mogu reći da se uobičajena mudrost koja kaže da je "imenovanje teško" primjenjuje i ovdje; vidio sam dramatično različite rezultate od VJM-ova samo promjenom naziva funkcija ili formulacije opisa. Ponekad uklanjanje opisa *poboljšava* performanse.

Obrada rezultata alata

Prilikom prosljeđivanja rezultata alata natrag VJM-u, osigurajte da su dobro strukturirani i sveobuhvatni. Koristite smislene ključeve i vrijednosti za predstavljanje izlaza svakog alata. Eksperimentirajte s različitim formatima i vidite koji najbolje funkcionira, od JSON-a do običnog teksta.

Interpreter rezultata rješava ovaj izazov korištenjem AI-ja za analizu rezultata i pružanje objašnjenja, sažetaka ili ključnih zaključaka prilagođenih ljudima.

Upravljanje pogreškama

Implementirajte robusne mehanizme za upravljanje pogreškama kako biste obradili slučajeve kada VJM može generirati nevažeće ili nepodržane ulazne parametre za pozive

alata. Elegantno upravljajte i oporavite se od bilo kakvih pogrešaka koje se mogu pojaviti tijekom izvršavanja alata.

Jedna izuzetno lijepa kvaliteta AI-ja je da razumije poruke o pogreškama! Što znači da ako radite u brzom i površnom načinu razmišljanja, možete jednostavno uhvatiti sve iznimke generirane u implementaciji alata i proslijediti ih natrag AI-ju kako bi znao što se dogodilo!

Na primjer, evo pojednostavljene verzije implementacije Google pretraživanja u Olympiji:

```
1   def google_search(conversation, params)
2     conversation.update_cstatus("Searching Google...")
3     query = params[:query]
4     search = GoogleSearch.new(query).get_hash
5
6     conversation.update_cstatus("Summarizing results...")
7     SummarizeKnowledgeGraph.new.perform(conversation, search.to_json)
8   rescue StandardError => e
9     Honeybadger.notify(e)
10    { error: e.message }.inspect
11  end
```

Google pretraživanja u Olympiji su dvostupanjski proces. Prvo se izvrši pretraživanje, zatim se sažmu rezultati. Ako dođe do bilo kakve pogreške, bez obzira kakve, poruka o pogrešci se zapakira i šalje natrag AI-ju. Ova tehnika je temelj praktički svih obrazaca *Inteligentnog rukovanja pogreškama.*

Na primjer, recimo da `GoogleSearch` API poziv ne uspije zbog 503 Service Unavailable iznimke. Ta se pogreška propagira do najviše razine hvatanja iznimki, a opis pogreške se šalje natrag AI-ju kao rezultat funkcijskog poziva. Umjesto da korisniku prikaže prazan zaslon ili tehničku pogrešku, AI kaže nešto poput "Žao mi je, ali trenutno ne mogu pristupiti svojim mogućnostima Google pretraživanja. Mogu pokušati kasnije, ako želite."

Ovo se može činiti kao samo domišljat trik, ali razmotrite drugačiju vrstu pogreške, onu

gdje AI poziva vanjski API i ima izravnu kontrolu nad parametrima koje šalje API-ju. Možda je pogriješio u načinu na koji je generirao te parametre? Pod uvjetom da je poruka o pogrešci vanjskog API-ja dovoljno detaljna, prosljeđivanje poruke o pogrešci natrag AI-ju koji je izvršio poziv znači da može preispitati te parametre i pokušati ponovno. Automatski. Bez obzira na to kakva je pogreška bila.

Sada razmislite što bi bilo potrebno da se replicira takva vrsta robusnog rukovanja pogreškama u *normalnom* kodu. To je praktički nemoguće.

Iterativno poboljšanje

Ako LLM ne preporučuje odgovarajuće alate ili generira suboptimalne odgovore, iterirajte kroz definicije alata, opise i ulazne parametre. Kontinuirano dorađujte i poboljšavajte postavke alata na temelju uočenog ponašanja i željenih ishoda.

1. Započnite s jednostavnim definicijama alata: Počnite definiranjem alata s jasnim i sažetim nazivima, opisima i ulaznim parametrima. U početku izbjegavajte prekomplicirano postavljanje alata i usredotočite se na osnovnu funkcionalnost. Na primjer, ako želite spremiti rezultate analize sentimenta, počnite s 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 promatrajte: Nakon što postavite početne definicije alata, testirajte ih različitim upitima i promatrajte kako VJM komunicira s alatom. Obratite pozornost na kvalitetu i relevantnost generiranih odgovora. Ako VJM generira suboptimalne odgovore, vrijeme je za usavršavanje definicija alata.
3. Usavršite opise: Ako VJM pogrešno razumije svrhu alata, pokušajte usavršiti opis alata. Pružite više konteksta, primjera ili pojašnjenja kako biste usmjerili VJM u učinkovito korištenje alata. Na primjer, 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 generira nevažeće ili irelevantne ulazne parametre za alat, razmislite o prilagodbi definicija parametara. Dodajte specifičnija ograničenja, pravila validacije ili primjere kako biste pojasnili očekivani format unosa.

5. Iteriranje na temelju povratnih informacija: Kontinuirano pratite performanse svojih alata i prikupljajte povratne informacije od korisnika ili dionika. Koristite te povratne informacije za identifikaciju područja koja treba poboljšati i napravite iterativna poboljšanja definicija alata. Na primjer, 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 temelju promatranog ponašanja i povratnih informacija, možete postupno poboljšati performanse i učinkovitost vaše aplikacije temeljene na umjetnoj inteligenciji. Ne zaboravite održavati definicije alata jasnima, sažetima i usredotočenima na specifični zadatak. Redovito testirajte i provjeravajte interakcije alata kako biste osigurali da su u skladu s željenim ishodima.

Sastavljanje i ulančavanje alata

Jedan od najmoćnijih aspekata korištenja alata, koji je do sada samo spomenut, jest mogućnost sastavljanja i ulančavanja više alata zajedno za izvršavanje složenih zadataka. Pažljivim dizajniranjem definicija alata i njihovih ulaznih/izlaznih formata možete stvoriti ponovno upotrebljive građevne blokove koji se mogu kombinirati na različite načine.

Razmotrimo primjer gdje gradite protok analize podataka za vašu aplikaciju temeljenu na umjetnoj inteligenciji. Mogli biste imati sljedeće alate:

1. **`DataRetrieval`**: Alat koji dohvaća podatke iz baze podataka ili API-ja na temelju određenih kriterija.

2. **`DataProcessing`**: Alat koji izvodi izračune, transformacije ili agregacije na dohvaćenim podacima.
3. **`DataVisualization`**: Alat koji predstavlja obrađene podatke u formatu prilagođenom korisniku, poput grafikona ili dijagrama.

Ulančavanjem ovih alata možete stvoriti moćan tijek rada koji dohvaća relevantne podatke, obrađuje ih i predstavlja rezultate na smislen način. Evo kako bi tijek rada s alatima mogao izgledati:

1. LLM prima korisnički upit koji traži uvide o podacima o prodaji za određenu kategoriju proizvoda.
2. LLM odabire alat `DataRetrieval` i generira odgovarajuće ulazne parametre za dohvaćanje relevantnih podataka o prodaji iz baze podataka.
3. Dohvaćeni podaci se "prosljeđuju" alatu `DataProcessing`, koji izračunava metrike poput ukupnog prihoda, prosječne prodajne cijene i stope rasta.
4. Obrađene podatke zatim koristi alat `DataVisualization`, koji stvara vizualno privlačan grafikon ili dijagram za prikaz uvida, vraćajući URL grafikona natrag LLM-u.
5. Na kraju, LLM generira formatirani odgovor na korisnički upit koristeći markdown, uključujući vizualizirane podatke i sažetak ključnih nalaza.

Sastavljanjem ovih alata zajedno možete stvoriti besprijekoran tijek rada analize podataka koji se može lako integrirati u vašu aplikaciju. Ljepota ovog pristupa je u tome što se svaki alat može razvijati i testirati neovisno, a zatim kombinirati na različite načine za rješavanje različitih problema.

Za omogućavanje glatkog sastavljanja i ulančavanja alata, važno je definirati jasne ulazne i izlazne formate za svaki alat.

Na primjer, alat `DataRetrieval` mogao bi prihvaćati parametre poput detalja povezivanja s bazom podataka, imena tablice i uvjeta upita, te vratiti skup rezultata kao

strukturirani JSON objekt. Alat `DataProcessing` tada može očekivati taj JSON objekt kao ulaz i proizvesti transformirani JSON objekt kao izlaz. Standardiziranjem protoka podataka između alata možete osigurati kompatibilnost i mogućnost ponovne upotrebe.

Dok dizajnirate svoj ekosustav alata, razmislite o tome kako se različiti alati mogu kombinirati za rješavanje uobičajenih slučajeva upotrebe u vašoj aplikaciji. Razmotrite stvaranje alata visoke razine koji obuhvaćaju uobičajene tijekove rada ili poslovnu logiku, čineći LLM-u lakšim njihov odabir i učinkovito korištenje.

Zapamtite, snaga korištenja alata leži u fleksibilnosti i modularnosti koju pruža. Raščlanjivanjem složenih zadataka na manje, ponovno upotrebljive alate, možete stvoriti robusnu i prilagodljivu aplikaciju temeljenu na umjetnoj inteligenciji koja može riješiti širok raspon izazova.

Budući smjerovi

Kako se područje razvoja aplikacija temeljenih na umjetnoj inteligenciji razvija, možemo očekivati daljnje napretke u mogućnostima korištenja alata. Neki potencijalni budući smjerovi uključuju:

1. **Višestruko korištenje alata**: LLM-ovi bi mogli biti u stanju odlučiti koliko puta trebaju koristiti alate kako bi generirali zadovoljavajući odgovor. To bi moglo uključivati više krugova odabira i izvršavanja alata na temelju međurezultata.
2. **Unaprijed definirani alati**: AI platforme mogle bi pružiti set unaprijed definiranih alata koje programeri mogu koristiti odmah, poput Python interpretera, alata za pretraživanje weba ili uobičajenih pomoćnih funkcija.
3. **Besprijekorna integracija**: Kako korištenje alata postaje sve raširenije, možemo očekivati bolju integraciju između AI platformi i popularnih razvojnih okvira, olakšavajući programerima uključivanje korištenja alata u njihove aplikacije.

Korištenje alata je moćna tehnika koja omogućuje programerima da iskoriste puni potencijal LLM-ova u aplikacijama temeljenima na umjetnoj inteligenciji. Povezivanjem LLM-ova s vanjskim alatima i resursima možete stvoriti dinamičnije, inteligentnije i kontekstualno svjesnije sustave koji se mogu prilagoditi potrebama korisnika i pružiti vrijedne uvide i akcije.

Iako korištenje alata nudi ogromne mogućnosti, važno je biti svjestan potencijalnih izazova i razmatranja. Jedan ključni aspekt je upravljanje složenošću interakcija alata i osiguravanje stabilnosti i pouzdanosti cjelokupnog sustava. Morate upravljati scenarijima gdje pozivi alata mogu ne uspjeti, vratiti neočekivane rezultate ili imati implikacije na performanse. Dodatno, trebali biste razmotriti sigurnosne mjere i kontrolu pristupa kako biste spriječili neovlašteno ili zlonamjerno korištenje alata. Pravilno rukovanje pogreškama, bilježenje i mehanizmi praćenja ključni su za održavanje integriteta i performansi vaše aplikacije temeljene na umjetnoj inteligenciji.

Dok istražujete mogućnosti upotrebe alata u svojim projektima, zapamtite da trebate početi s jasnim ciljevima, dizajnirati dobro strukturirane definicije alata i iterirati na temelju povratnih informacija i rezultata. Uz pravi pristup i način razmišljanja, upotreba alata može otključati nove razine inovacija i vrijednosti u vašim aplikacijama temeljenima na umjetnoj inteligenciji

Obrada toka podataka

Prijenos podataka preko HTTP-a, poznat i kao eventi poslani od strane poslužitelja (SSE), mehanizam je kojim poslužitelj kontinuirano šalje podatke klijentu čim postanu dostupni, bez potrebe da ih klijent eksplicitno zatraži. Budući da se AI-jev odgovor generira postupno, ima smisla pružiti responzivno korisničko iskustvo prikazivanjem AI-jevog izlaza tijekom njegovog generiranja. Zapravo, svi API-ji pružatelja AI usluga koje poznajem nude mogućnost prijenosa odgovora kao opciju u svojim završnim točkama.

Razlog zašto se ovo poglavlje pojavljuje ovdje u knjizi, odmah nakon Korištenje alata, jest zbog toga koliko moćno može biti kombiniranje korištenja alata sa živim AI odgovorima korisnicima. To omogućuje dinamična i interaktivna iskustva gdje AI može obrađivati korisnički unos, koristiti različite alate i funkcije po vlastitom nahođenju te pružati odgovore u stvarnom vremenu.

Za postizanje ove besprijekorne interakcije, potrebno je napisati rukovatelje toka podataka koji mogu otpremati pozive funkcija alata koje poziva AI, kao i običan

tekstualni izlaz krajnjem korisniku. Potreba za ponavljanjem petlje nakon obrade funkcije alata dodaje zanimljiv izazov zadatku.

Implementacija ReplyStream-a

Kako bi se pokazalo kako se može implementirati obrada toka podataka, ovo poglavlje će detaljno obraditi pojednostavljenu verziju klase `ReplyStream` koja se koristi u Olympiji. Instance ove klase mogu se proslijediti kao parametar `stream` u AI klijentskim bibliotekama kao što su ruby-openai i openrouter

Evo kako koristim `ReplyStream` u Olympijinom `PromptSubscriber`-u, koji putem Wisper-a osluškuje stvaranje novih korisničkih poruka.

```
class PromptSubscriber
  include Raix::ChatCompletion
  include Raix::PromptDeclarations

  # many other declarations omitted...

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

  def message_created(message) # invoked by Wisper
    return unless message.role.user? && message.content?

    # rest of the implementation omitted...
```

Uz referencu `context` na pretplatnika na upit koji ga je instancirao, klasa `ReplyStream` također ima varijable instance za pohranu međuspremnika primljenih podataka te polja za praćenje imena funkcija i argumenata koji se pozivaju tijekom obrade toka.

```ruby
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`, inicijalizirajući međuspremnik, kontekst i druge varijable.

Metoda `call` je glavna ulazna točka za obradu podataka koji se strujaju. Prima `chunk` podataka (predstavljen kao hash) i opcionalni parametar `bytesize`, koji u našem primjeru nije korišten. Unutar ove metode, klasa koristi podudaranje uzoraka za rukovanje različitim scenarijima temeljenim na strukturi primljenog bloka podataka.

Pozivanje `deep_symbolize_keys` na bloku podataka čini podudaranje uzoraka elegantnijim, omogućujući nam rad sa simbolima umjesto 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 uzorak koji tražimo je poziv alata zajedno s pripadajućim nazivom funkcije. Ako ga otkrijemo, spremamo ga u polje `f_name`. Nazive funkcija pohranjujemo u indeksirano polje jer model može paralelno pozivati funkcije, šaljući više funkcija na izvršavanje istovremeno.

> Paralelno pozivanje funkcija je sposobnost AI modela da izvršava više poziva funkcija zajedno, omogućujući da se učinci i rezultati tih poziva funkcija rješavaju paralelno. Ovo je posebno korisno ako funkcije dugo traju i smanjuje broj povratnih putovanja s API-jem, što zauzvrat može značajno smanjiti potrošnju tokena.

Zatim trebamo 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 s imenima funkcija, argumente spremamo u indeksirano polje.

Zatim, pratimo obične poruke namijenjene korisniku koje će stizati s poslužitelja token po token i biti dodijeljene varijabli `new_content`. Također moramo paziti na `finish_reason`. On će biti `nil` sve do posljednjeg fragmenta izlaznog niza.

```
  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 napomenuti da dodajemo izraz za podudaranje uzoraka kako bismo obradili poruke o pogreškama koje šalje pružatelj AI modela. U lokalnim razvojnim okruženjima, podižemo iznimku, ali u produkciji pogrešku bilježimo i finaliziramo.

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

Završna else klauzula case naredbe izvršit će se ako nijedan od prethodnih uzoraka nije odgovarao. To je samo zaštitna mjera kako bismo saznali ako AI model počne slati neprepoznate dijelove.

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

Metoda `send_to_client` zadužena je za slanje međuspremljenog sadržaja klijentu. Ona provjerava je li međuspremnik prazan, ažurira sadržaj poruke bota, prikazuje poruku bota te sprema sadržaj u bazu 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
```

Metoda `finalize` poziva se kada je obrada toka završena. Ona izvršava pozive funkcija ako su neki primljeni tijekom toka, ažurira poruku bota s konačnim sadržajem i drugim relevantnim informacijama te resetira povijest 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 pozvati funkciju, morate "otpremiti" taj poziv funkcije (naziv i argumente) na način da se izvrši i da se poruke `function_call` i `function_result` dodaju u transkript razgovora

Prema mom iskustvu, bolje je upravljati stvaranjem funkcijskih poruka na jednom mjestu u vašoj bazi koda, umjesto da se oslanjate na implementacije alata. To nije samo urednije, već ima i vrlo važan praktični razlog: ako AI model pozove funkciju, a ne vidi rezultirajuće poruke poziva i rezultata u transkriptu kada se petlja ponovi, *pozvat će istu funkciju ponovno*. Potencijalno beskonačno. Zapamtite da je AI potpuno bez stanja, pa ako ne prikažete te pozive funkcija natrag 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 povijesti poziva funkcija nakon izvršavanja jednako je važno kao i osiguravanje da se poziv i rezultati pojave u vašem transkriptu, kako ne biste neprestano pozivali iste funkcije iznova i iznova svaki put kad se petlja izvršava.

"Konverzacijska petlja"

Ja stalno spominjem petlje, ali ako ste novi u pozivanju funkcija, možda nije očito *zašto* nam treba petlja. Razlog je taj što kada AI "zatraži" da izvršite pomoćne funkcije u njegovo ime, prestat će odgovarati. Na vama je da izvršite te funkcije, prikupite rezultate, dodate rezultate u transkript i zatim ponovno pošaljete izvorni upit kako biste dobili novi skup poziva funkcija ili rezultata vidljivih korisniku.

U klasi `PromptSubscriber`, koristimo metodu `prompt` iz modula `PromptDeclarations` za definiranje ponašanja konverzacijske petlje. Parametar `until` postavljen je na `-> { bot_message.complete? }`, što znači da će se petlja nastaviti izvršavati sve dok `bot_message` nije označen kao završen.

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

Ali kada je `bot_message` označen kao dovršen? Ako ste zaboravili, pogledajte redak 13 metode `finalize`.

Pregledajmo cijelu logiku obrade toka podataka.

1. `PromptSubscriber` prima novu korisničku poruku putem metode `message_created`, koju poziva Wisper sustav objavi/pretplati svaki put kada krajnji korisnik kreira novi upit.
2. Klasna metoda `prompt` deklarativno definira ponašanje logike dovršavanja razgovora za `PromptSubscriber`. AI model će izvršiti dovršavanje razgovora s korisničkim sadržajem poruke, novom instancom `ReplyStream` kao parametrom toka i određenim uvjetom petlje.
3. AI model obrađuje upit i počinje generirati odgovor. Kako se odgovor prenosi u toku, metoda `call` instance `ReplyStream` poziva se za svaki dio podataka.
4. Ako AI model odluči pozvati pomoćnu funkciju, naziv funkcije i argumenti se izdvajaju iz dijela podataka i pohranjuju u nizove `f_name` i `f_arguments`.
5. Ako AI model generira sadržaj vidljiv korisniku, on se privremeno pohranjuje i šalje klijentu putem metode `send_to_client`.
6. Nakon što je obrada toka završena, poziva se metoda `finalize`. Ako su tijekom toka pozvane bilo koje pomoćne funkcije, one se izvršavaju pomoću metode `dispatch` klase `PromptSubscriber`.
7. Metoda `dispatch` dodaje poruku `function_call` u transkript razgovora, izvršava odgovarajuću pomoćnu funkciju i dodaje poruku `function_result` u transkript s rezultatom poziva funkcije.
8. Nakon izvršavanja pomoćnih funkcija, povijest poziva funkcija se briše kako bi se spriječili duplicirani pozivi funkcija u sljedećim iteracijama.

9. Ako nisu pozvane pomoćne funkcije, metoda `finalize` ažurira `bot_message` s konačnim sadržajem, označava ga kao dovršenog i šalje ažuriranu poruku klijentu.
10. Uvjet petlje `-> { bot_message.complete? }` se evaluira. Ako `bot_message` nije označen kao dovršen, petlja se nastavlja, a izvorni upit se ponovno šalje s ažuriranim transkriptom razgovora.
11. Koraci 3-10 se ponavljaju dok `bot_message` nije označen kao dovršen, što označava da je AI model završio generiranje svog odgovora i da nema potrebe za izvršavanjem dodatnih pomoćnih funkcija.

Implementacijom ove konverzacijske petlje, omogućujete AI modelu da se upusti u dvosmjernu interakciju s aplikacijom, izvršavajući pomoćne funkcije prema potrebi i generirajući odgovore vidljive korisniku sve dok razgovor ne dođe do prirodnog zaključka.

Kombinacija obrade toka podataka i konverzacijske petlje omogućuje dinamična i interaktivna AI iskustva, gdje AI model može obrađivati korisničke unose, koristiti različite alate i funkcije te pružati odgovore u stvarnom vremenu na temelju razvijajućeg konteksta razgovora.

Automatski nastavak

Važno je biti svjestan ograničenja AI izlaza. Većina modela ima maksimalni broj tokena koje mogu generirati u jednom odgovoru, što je određeno parametrom `max_tokens`. Ako AI model dosegne ovo ograničenje tijekom generiranja odgovora, naglo će se zaustaviti i naznačiti da je izlaz skraćen.

U odgovoru koji se prenosi tokom s API-ja AI platforme, ovu situaciju možete otkriti pregledavanjem varijable `finish_reason` u dijelu podataka. Ako je `finish_reason` postavljen na `"length"` (ili neku drugu ključnu vrijednost specifičnu za model), to znači da je model dosegao svoje maksimalno ograničenje tokena tijekom generiranja i da je izlaz prijevremeno prekinut.

Jedan od načina da se ova situacija elegantno riješi i pruži besprijekorno korisničko iskustvo jest implementacija mehanizma automatskog nastavka u logici obrade toka. Dodavanjem uzorka podudaranja za razloge završetka povezane s duljinom, možete odabrati ponavljanje petlje i automatski nastaviti izlaz od mjesta gdje je prekinut.

Evo namjerno pojednostavljenog primjera kako možete modificirati 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 modificiranoj verziji, kada `finish_reason` ukazuje na odrezani izlaz, umjesto završavanja toka, dodajemo par poruka u transkript bez finaliziranja, premještamo izvornu poruku namijenjenu korisniku na "dno" transkripta ažuriranjem njenog atributa `created_at`, a zatim dopuštamo petlji da se nastavi, tako da UI nastavlja generirati gdje je stao.

Zapamtite da je krajnja točka UI završetka bez stanja. Ona "zna" samo ono što joj kažete putem transkripta. U ovom slučaju, način na koji UI-u komuniciramo da je bio prekinut jest dodavanjem "nevidljivih" (krajnjem korisniku) poruka u transkript. Međutim, zapamtite da je ovo namjerno pojednostavljeni primjer. Stvarna implementacija bi trebala provesti dodatno upravljanje transkriptom kako bi se osiguralo da ne trošimo tokene i/ili ne zbunimo UI dupliciranim porukama asistenta u transkriptu.

Stvarna implementacija automatskog nastavljanja također bi trebala imati takozvanu "logiku prekidača" kako bi se spriječilo nekontrolirano ponavljanje petlje. Razlog tome je što bi, s određenim vrstama korisničkih upita i niskim postavkama `max_tokens`, UI mogao beskonačno ponavljati izlaz namijenjen korisniku.

> Imajte na umu da svaka petlja zahtijeva zasebni zahtjev i da svaki zahtjev ponovno koristi cijeli vaš transkript. Svakako biste trebali razmotriti kompromise između korisničkog iskustva i korištenja API-ja kada odlučujete hoćete li implementirati automatsko nastavljanje u svojoj aplikaciji. Automatsko nastavljanje može biti posebno opasno skupo, osobito pri korištenju premium komercijalnih modela.

Zaključak

Obrada toka podataka ključni je aspekt izgradnje aplikacija pokretanih umjetnom inteligencijom koje kombiniraju korištenje alata s UI odgovorima u stvarnom vremenu. Učinkovitim upravljanjem tokovima podataka iz API-ja UI platformi možete pružiti

besprijekorno i interaktivno korisničko iskustvo, rukovati velikim odgovorima, optimizirati korištenje resursa i elegantno upravljati pogreškama.

Predstavljena klasa `Conversation::ReplyStream` pokazuje kako se obrada toka podataka može implementirati u Ruby aplikaciji koristeći podudaranje uzoraka i arhitekturu vođenu događajima. Razumijevanjem i korištenjem tehnika obrade toka podataka možete otključati puni potencijal UI integracije u svojim aplikacijama i pružiti moćna i privlačna korisnička iskustva.

Samozacjeljujući podaci

Samozacjeljujući podaci su moćan pristup osiguravanju integriteta, konzistentnosti i kvalitete podataka u aplikacijama korištenjem mogućnosti velikih jezičnih modela (LLM-ova). Ova kategorija obrazaca fokusira se na ideju korištenja UI-ja za automatsko otkrivanje, dijagnosticiranje i ispravljanje anomalija, nekonzistentnosti ili pogrešaka u podacima, čime se smanjuje opterećenje razvojnih inženjera i održava visoka razina pouzdanosti podataka.

U svojoj srži, obrasci samozacjeljujućih podataka prepoznaju da su podaci životna snaga svake aplikacije, a osiguravanje njihove točnosti i integriteta ključno je za pravilno funkcioniranje i korisničko iskustvo aplikacije. Međutim, upravljanje i održavanje kvalitete podataka može biti složen i vremenski zahtjevan zadatak, posebno kako aplikacije rastu u veličini i složenosti. Tu na scenu stupa snaga umjetne inteligencije.

U obrascima samozacjeljujućih podataka, UI radnici se koriste za kontinuirano praćenje i analizu podataka vaše aplikacije. Ovi modeli imaju sposobnost razumijevanja i

interpretacije obrazaca, odnosa i anomalija unutar podataka. Koristeći svoje sposobnosti obrade i razumijevanja prirodnog jezika, mogu identificirati potencijalne probleme ili nekonzistentnosti u podacima i poduzeti odgovarajuće radnje za njihovo ispravljanje.

Proces samozacjeljivanja podataka obično uključuje nekoliko ključnih koraka:

1. **Praćenje podataka**: UI radnici konstantno prate tokove podataka aplikacije, baze podataka ili sustave za pohranu, tražeći bilo kakve znakove anomalija, nekonzistentnosti ili pogrešaka. Alternativno, možete aktivirati UI komponentu kao reakciju na iznimku.
2. **Otkrivanje anomalija**: Kada se otkrije problem, UI radnik detaljno analizira podatke kako bi identificirao specifičnu prirodu i opseg problema. To može uključivati otkrivanje nedostajućih vrijednosti, nekonzistentnih formata ili podataka koji krše preddefinirane pravila ili ograničenja.
3. **Dijagnoza i korekcija**: Nakon što je problem identificiran, UI radnik koristi svoje znanje i razumijevanje domene podataka kako bi odredio odgovarajući smjer djelovanja. To može uključivati automatsko ispravljanje podataka, popunjavanje nedostajućih vrijednosti ili označavanje problema za ljudsku intervenciju ako je potrebno.
4. **Kontinuirano učenje (opcionalno, ovisno o slučaju uporabe)**: Kako vaš UI radnik susreće i rješava različite probleme s podacima, može generirati opise onoga što se dogodilo i kako je reagirao. Ti metapodaci mogu se unijeti u procese učenja koji vam (i možda osnovnom modelu, putem finog podešavanja) omogućuju da postanete učinkovitiji tijekom vremena u identificiranju i rješavanju anomalija podataka.

Automatskim otkrivanjem i ispravljanjem problema s podacima možete osigurati da vaša aplikacija radi s visokokvalitetnim, pouzdanim podacima. Time se smanjuje rizik od pogrešaka, nekonzistentnosti ili bugova povezanih s podacima koji utječu na funkcionalnost aplikacije ili korisničko iskustvo.

Jednom kada UI radnici preuzmu zadatak praćenja i ispravljanja podataka, možete usmjeriti svoje napore na druge kritične aspekte aplikacije. Time se štedi vrijeme i resursi 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 "Samozacjeljujućih podataka" učinkovito se skaliraju koristeći snagu UI-ja za obradu velikih količina podataka i otkrivanje problema u stvarnom vremenu.

Zbog svoje prirode, UI modeli se mogu prilagoditi promjenjivim obrascima podataka, shemama ili zahtjevima tijekom vremena s malo ili bez nadzora. Sve dok njihove direktive pružaju adekvatne smjernice, posebno u pogledu željenih rezultata, vaša aplikacija može evoluirati i upravljati novim scenarijima podataka bez potrebe za opsežnom ručnom intervencijom ili promjenama koda.

Obrasci samozacjeljujućih podataka dobro se usklađuju s drugim kategorijama obrazaca o kojima smo raspravljali, poput "Mnoštva radnika". Mogućnost samozacjeljivanja podataka može se promatrati kao specijalizirana vrsta radnika koji se posebno fokusira na osiguravanje kvalitete i integriteta podataka. Ova vrsta radnika djeluje zajedno s drugim UI radnicima, pri čemu svaki doprinosi različitim aspektima funkcionalnosti aplikacije.

Implementacija obrazaca samozacjeljujućih podataka u praksi zahtijeva pažljivo projektiranje i integraciju UI modela u arhitekturu aplikacije. Zbog rizika od gubitka i oštećenja podataka, trebali biste definirati jasne smjernice za način na koji ćete koristiti ovu tehniku. Također biste trebali uzeti u obzir čimbenike kao što su performanse, skalabilnost i sigurnost podataka.

Praktični primjer: Popravljanje neispravnog JSON-a

Jedan od najpraktičnijih i najjednostavnijih načina korištenja samozacjeljujućih podataka također je vrlo jednostavan za objašnjenje: popravljanje neispravnog JSON-a.

Ova tehnika može se primijeniti na uobičajeni izazov rada s nesavršenim ili nekonzistentnim podacima koje generiraju LLM-ovi, poput neispravnog JSON-a, i pruža pristup za automatsko otkrivanje i ispravljanje tih problema.

U Olympiji redovito se susrećem sa scenarijima gdje LLM-ovi generiraju JSON podatke koji nisu potpuno ispravni. To se može dogoditi iz različitih razloga, kao što su slučajevi kada LLM dodaje komentare prije ili nakon stvarnog JSON koda, ili uvodi sintaksne pogreške poput nedostajućih zareza ili neescapeanih dvostrukih navodnika. Ovi problemi mogu dovesti do grešaka parsiranja i uzrokovati prekide u funkcionalnosti aplikacije.

Kako bih riješio ovaj problem, implementirao sam praktično rješenje u obliku JsonFixer klase. Ova klasa utjelovljuje uzorak "Samozacjeljujućih podataka" tako što uzima neispravan JSON kao ulaz i koristi LLM za njegovo popravljanje, pritom čuvajući što je moguće više informacija i izvornu namjeru.

```
1  class JsonFixer
2    include Raix::ChatCompletion
3
4    def call(bad_json, error_message)
5      raise "No data provided" if bad_json.blank? || error_message.blank?
6
7      transcript << {
8          system: "Consider user-provided JSON that generated a parse
9                   exception. Do your best to fix it while preserving the
10                  original content and intent as much as possible." }
11     transcript << { user: bad_json }
12     transcript << { assistant: "What is the error message?"}
13     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
end
```

Primijetite kako `JsonFixer` koristi Ventriloquist za usmjeravanje AI odgovora.

Proces samooporavljanja JSON podataka funkcionira na sljedeći način:

1. **Generiranje JSON-a**: LLM se koristi za generiranje JSON podataka na temelju određenih upita ili zahtjeva. Međutim, zbog prirode LLM-ova, generirani JSON nije uvijek savršeno valjan. JSON parser će, naravno, izbaciti `ParserError` ako mu date nevaljani JSON.

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

Imajte na umu da se poruka o pogrešci također prosljeđuje pozivu `JSONFixer`-a tako da ne mora u potpunosti pretpostavljati što nije u redu s podacima, posebno zato što će parser često točno reći što nije u redu.

2. **Ispravak temeljen na LLM-u**: Klasa `JSONFixer` šalje neispravan JSON natrag LLM-u, zajedno s određenim upitom ili uputom za popravak JSON-a uz

maksimalno očuvanje izvornih informacija i namjere. LLM, treniran na ogromnoj količini podataka i s razumijevanjem JSON sintakse, pokušava ispraviti pogreške i generirati valjani JSON string. Ograđivanje odgovora koristi se za ograničavanje izlaza LLM-a, a odabiremo Mixtral 8x7B kao AI model, jer je posebno dobar za ovu vrstu zadatka.

3. **Validacija i integracija**: Popravljeni JSON string koji vraća LLM parsira sama klasa `JSONFixer`, jer je pozvala `chat_completion(json: true)`. Ako popravljeni JSON prođe validaciju, integrira se natrag u tijek rada aplikacije, omogućujući aplikaciji da nastavi obrađivati podatke bez prekida. Neispravan JSON je "izliječen".

Iako sam napisao i prepisao vlastitu implementaciju `JSONFixer`-a nekoliko puta, sumnjam da je ukupno vrijeme uloženo u sve te verzije više od sat ili dva.

Imajte na umu da je očuvanje namjere ključni element svakog uzorka samozacjeljujućih podataka. Proces ispravka temeljen na LLM-u nastoji maksimalno očuvati izvorne informacije i namjeru generiranog JSON-a. To osigurava da popravljeni JSON zadržava svoje semantičko značenje i može se učinkovito koristiti unutar konteksta aplikacije.

Ova praktična implementacija pristupa "Samozacjeljujućih podataka" u Olympiji jasno pokazuje kako se AI, posebno LLM-ovi, mogu iskoristiti za rješavanje stvarnih podatkovnih izazova. Pokazuje snagu kombiniranja tradicionalnih programerskih tehnika s AI mogućnostima za izgradnju robusnih i učinkovitih aplikacija.

Postelov zakon i uzorak "Samozacjeljujućih podataka"

"Samozacjeljujući podaci", kao što je prikazano klasom JSONFixer, dobro se usklađuju s principom poznatim kao Postelov zakon, koji se također naziva Princip robusnosti.

Postelov zakon glasi:

"Budite konzervativni u onome što činite, budite liberalni u onome što prihvaćate od drugih."

Ovaj princip, koji je izvorno artikulirao Jon Postel, pionir ranog Interneta, naglašava važnost izgradnje sustava koji su tolerantni prema raznolikim ili čak blago netočnim ulazima, dok istovremeno održavaju strogo pridržavanje specificiranih protokola pri slanju izlaza.

U kontekstu "Samozacjeljujućih podataka", klasa JSONFixer utjelovljuje Postelov zakon time što je liberalna u prihvaćanju neispravnih ili nesavršenih JSON podataka generiranih od strane LLM-ova. Ne odbacuje odmah i ne pada kada naiđe na JSON koji se strogo ne pridržava očekivanog formata. Umjesto toga, zauzima tolerantan pristup i pokušava popraviti JSON koristeći snagu LLM-ova.

Budući da je liberalna u prihvaćanju nesavršenog JSON-a, klasa JSONFixer pokazuje robusnost i fleksibilnost. Priznaje da podaci u stvarnom svijetu često dolaze u različitim oblicima i ne moraju se uvijek pridržavati strogih specifikacija. Elegantnim rukovanjem i ispravljanjem ovih odstupanja, klasa osigurava da aplikacija može nastaviti glatko funkcionirati, čak i u prisutnosti nesavršenih podataka.

S druge strane, klasa JSONFixer se također pridržava konzervativnog aspekta Postelovog zakona kada je riječ o izlazu. Nakon popravljanja JSON-a pomoću LLM-ova, klasa validira ispravljeni JSON kako bi osigurala da se strogo pridržava očekivanog formata. Održava integritet i ispravnost podataka prije nego što ih proslijedi drugim dijelovima aplikacije. Ovaj konzervativni pristup jamči da je izlaz klase JSONFixer pouzdan i konzistentan, promičući interoperabilnost i sprječavajući širenje pogrešaka.

Zanimljivosti o Jonu Postelu:

- Jon Postel (1943-1998) bio je američki računalni znanstvenik koji je odigrao ključnu ulogu u razvoju Interneta. Bio je poznat kao "Bog Interneta" zbog

svojih značajnih doprinosa temeljnim protokolima i standardima.

- Postel je bio urednik serije dokumenata Request for Comments (RFC), što je serija tehničkih i organizacijskih bilješki o Internetu. Autor je ili koautor više od 200 RFC-ova, uključujući temeljne protokole kao što su TCP, IP i SMTP.
- Uz svoje tehničke doprinose, Postel je bio poznat po svom skromnom i suradničkom pristupu. Vjerovao je u važnost postizanja konsenzusa i zajedničkog rada na izgradnji robusne i interoperabilne mreže.
- Postel je služio kao direktor Odjela za računalne mreže u Institutu za informacijske znanosti (ISI) Sveučilišta Južne Kalifornije (USC) od 1977. do svoje prerane smrti 1998.
- U znak priznanja za njegove ogromne doprinose, Postelu je posthumno dodijeljena prestižna Turingova nagrada 1998. godine, koja se često naziva "Nobelovom nagradom za računarstvo."

Klasa `JSONFixer` promiče robusnost, fleksibilnost i interoperabilnost, što su bile temeljne vrijednosti koje je Postel zastupao tijekom svoje karijere. Izgradnjom sustava koji su tolerantni na nesavršenosti, dok istovremeno strogo poštuju protokole, možemo stvoriti aplikacije koje su otpornije i prilagodljivije stvarnim izazovima.

Razmatranja i kontraindikacije

Primjenjivost pristupa samozacjeljujućih podataka u potpunosti ovisi o vrsti podataka kojima vaša aplikacija upravlja. Postoji razlog zašto možda ne želite jednostavno modificirati `JSON.parse` da automatski ispravlja *sve JSON pogreške parsiranja* u vašoj aplikaciji: nisu sve pogreške one koje se mogu ili trebaju automatski ispraviti.

Samozacjeljivanje je posebno problematično kada je povezano s regulatornim zahtjevima ili zahtjevima usklađenosti vezanim uz rukovanje i obradu podataka. Neke industrije, poput zdravstva i financija, imaju tako stroge propise u vezi s integritetom

podataka i mogućnošću revizije da bilo kakvo ispravljanje podataka po principu "crne kutije" bez pravilnog nadzora ili bilježenja može prekršiti te propise. Ključno je osigurati da sve tehnike samozacjeljivanja podataka koje osmislite budu usklađene s primjenjivim pravnim i regulatornim okvirima.

Primjena tehnika samozacjeljivanja podataka, posebno onih koje uključuju AI modele, također može imati velik utjecaj na performanse aplikacije i korištenje resursa. Obrada velikih količina podataka kroz AI modele za otkrivanje i ispravljanje pogrešaka može biti računalno zahtjevna. Važno je procijeniti kompromise između prednosti samozacjeljivanja podataka i povezanih troškova performansi i resursa.

To rečeno, zaronimo u čimbenike koji su uključeni u odlučivanje kada i gdje primijeniti ovaj moćni pristup.

Kritičnost podataka

Prilikom razmatranja primjene tehnika samozacjeljivanja podataka, ključno je procijeniti kritičnost podataka koji se obrađuju. Razina kritičnosti odnosi se na važnost i osjetljivost podataka u kontekstu vaše aplikacije i njezine poslovne domene.

U nekim slučajevima, automatsko ispravljanje pogrešaka u podacima možda nije prikladno, posebno ako su podaci vrlo osjetljivi ili imaju pravne implikacije. Na primjer, razmotrite sljedeće scenarije:

1. **Financijske transakcije**: U financijskim aplikacijama, poput bankarskih sustava ili trgovačkih platformi, točnost podataka je od najveće važnosti. Čak i male pogreške u financijskim podacima mogu imati značajne posljedice, poput netočnih stanja računa, pogrešno usmjerenih sredstava ili pogrešnih odluka o trgovanju. U tim slučajevima, automatske korekcije bez temeljite provjere i revizije mogu uvesti neprihvatljive rizike.
2. **Medicinski zapisi**: Zdravstvene aplikacije bave se vrlo osjetljivim i povjerljivim podacima pacijenata. Netočnosti u medicinskim zapisima mogu imati ozbiljne

implikacije za sigurnost pacijenata i odluke o liječenju. Automatsko modificiranje medicinskih podataka bez pravilnog nadzora i validacije od strane kvalificiranih zdravstvenih stručnjaka može prekršiti regulatorne zahtjeve i ugroziti dobrobit pacijenata.

3. **Pravni dokumenti**: Aplikacije koje rukuju pravnim dokumentima, poput ugovora, sporazuma ili sudskih podnesaka, zahtijevaju strogu točnost i integritet. Čak i male pogreške u pravnim podacima mogu imati značajne pravne posljedice. Automatske korekcije u ovom području možda nisu prikladne, jer podaci često zahtijevaju ručni pregled i verifikaciju od strane pravnih stručnjaka kako bi se osigurala njihova valjanost i provedivost.

U ovim kritičnim scenarijima podataka, rizici povezani s automatskim korekcijama često nadmašuju potencijalne koristi. Posljedice uvođenja pogrešaka ili neispravnog modificiranja podataka mogu biti ozbiljne, dovodeći do financijskih gubitaka, pravnih odgovornosti ili čak štete pojedincima.

Kada se radi o vrlo kritičnim podacima, ključno je dati prioritet procesima ručne verifikacije i validacije. Ljudski nadzor i stručnost su ključni u osiguravanju točnosti i integriteta podataka. Automatizirane tehnike samozacjeljivanja i dalje se mogu koristiti za označavanje potencijalnih pogrešaka ili nedosljednosti, ali konačna odluka o korekcijama trebala bi uključivati ljudsku prosudbu i odobrenje.

Međutim, važno je napomenuti da svi podaci u aplikaciji ne moraju imati istu razinu kritičnosti. Unutar iste aplikacije mogu postojati podskupovi podataka koji su manje osjetljivi ili imaju manji utjecaj ako dođe do pogrešaka. U takvim slučajevima, tehnike samozacjeljivanja podataka mogu se selektivno primijeniti na te specifične podskupove podataka, dok kritični podaci ostaju podložni ručnoj verifikaciji.

Ključno je pažljivo procijeniti kritičnost svake kategorije podataka u vašoj aplikaciji i definirati jasne smjernice i procese za rukovanje korekcijama na temelju povezanih rizika i implikacija. Razlikovanjem između kritičnih (npr. knjige računa, medicinski zapisi) i nekritičnih podataka (npr. poštanske adrese, upozorenja o resursima), možete

postići ravnotežu između iskorištavanja prednosti tehnika samozacjeljivanja podataka gdje je to prikladno i održavanja stroge kontrole i nadzora gdje je to potrebno.

U konačnici, odluka o primjeni tehnika samozacjeljivanja podataka na kritične podatke trebala bi se donijeti u konzultaciji sa stručnjacima iz domene, pravnim savjetnicima i drugim relevantnim dionicima. Ključno je razmotriti specifične zahtjeve, propise i rizike povezane s podacima vaše aplikacije i uskladiti strategije ispravljanja podataka u skladu s tim.

Ozbiljnost pogreške

Prilikom primjene tehnika samozacjeljivanja podataka, važno je procijeniti ozbiljnost i utjecaj pogrešaka u podacima. Nisu sve pogreške jednake, a odgovarajući postupak može varirati ovisno o ozbiljnosti problema.

Manje nedosljednosti ili problemi s formatiranjem mogu biti prikladni za automatsko ispravljanje. Na primjer, samozacjeljujući podatkovni radnik zadužen za popravljanje neispravnog JSON-a može obraditi nedostajuće zareze ili neeskeipirane dvostruke navodnike bez značajne promjene značenja ili strukture podataka. Ove vrste pogrešaka često je jednostavno ispraviti i imaju minimalan utjecaj na ukupni integritet podataka.

Međutim, ozbiljnije pogreške koje temeljno mijenjaju značenje ili integritet podataka mogu zahtijevati drugačiji pristup. U takvim slučajevima, automatizirani ispravci možda neće biti dovoljni, te će možda biti potrebna ljudska intervencija kako bi se osigurala točnost i valjanost podataka.

Tu na scenu stupa koncept korištenja same umjetne inteligencije za određivanje ozbiljnosti pogreške. Koristeći mogućnosti AI modela, možemo dizajnirati samoopravljajuće podatkovne radnike koji ne samo da ispravljaju pogreške, već i procjenjuju njihovu ozbiljnost te donose informirane odluke o načinu njihova rješavanja.

Na primjer, razmotrimo samoopravljajućeg podatkovnog radnika zaduženog za ispravljanje nedosljednosti u podacima koji se unose u bazu podataka kupaca. Radnik

može biti dizajniran da analizira podatke i identificira potencijalne pogreške, poput nedostajućih ili proturječnih informacija. Međutim, umjesto da automatski ispravlja sve pogreške, radnik može biti opremljen dodatnim pozivima alata koji mu omogućuju označavanje ozbiljnih pogrešaka za pregled od strane ljudi.

Evo primjera kako se to može implementirati:

```
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
```

```
35    end
36  end
```

U ovom primjeru, `CustomerDataHealer` radnik je dizajniran za prepoznavanje i ispravljanje nedosljednosti u podacima o kupcima. Još jednom koristimo ograđivanje odgovora i trbuhozborac za dobivanje strukturiranog izlaza. Važno je napomenuti da sistemska direktiva radnika uključuje upute za korištenje funkcije `flag_for_review` ako se naiđe na ozbiljne pogreške.

Kada radnik obrađuje podatke o kupcima, analizira podatke i pokušava ispraviti sve nedosljednosti. Ako radnik utvrdi da su pogreške ozbiljne i zahtijevaju ljudsku intervenciju, može koristiti alat `flag_for_review` za označavanje podataka i navođenje razloga za označavanje.

Metoda `chat_completion` poziva se s `json: true` kako bi se ispravljeni podaci o kupcu parsirali kao JSON. Ne postoji mogućnost petlje nakon poziva funkcije, pa će rezultat biti prazan ako je pozvana funkcija `flag_for_review`. U suprotnom, podaci o kupcu se ažuriraju s pregledanim i potencijalno ispravljenim podacima.

Uključivanjem procjene ozbiljnosti pogreške i mogućnosti označavanja podataka za ljudski pregled, samozacjeljujući radnik za podatke postaje inteligentniji i prilagodljiviji. Može automatski rješavati manje pogreške, dok ozbiljnije pogreške eskalira stručnjacima za ručnu intervenciju.

Specifični kriteriji za određivanje ozbiljnosti pogreške mogu se definirati u direktivi radnika na temelju domenskog znanja i poslovnih zahtjeva. Faktori poput utjecaja na integritet podataka, mogućnosti gubitka ili oštećenja podataka te posljedice netočnih podataka mogu se uzeti u obzir pri procjeni ozbiljnosti.

Korištenjem umjetne inteligencije za procjenu ozbiljnosti pogreške i pružanjem mogućnosti za ljudsku intervenciju, tehnike samozacjeljivanja podataka mogu postići ravnotežu između automatizacije i održavanja točnosti podataka. Ovaj pristup osigurava da se manje pogreške učinkovito ispravljaju, dok ozbiljne pogreške dobivaju potrebnu pažnju i stručnost od ljudskih recenzenata.

Složenost domene

Pri razmatranju primjene tehnika samozacjeljivanja podataka, važno je procijeniti složenost domene podataka i pravila koja upravljaju njihovom strukturom i odnosima. Složenost domene može značajno utjecati na učinkovitost i izvedivost automatiziranih pristupa ispravljanju podataka.

Tehnike samozacjeljivanja podataka dobro funkcioniraju kada podaci slijede dobro definirane obrasce i ograničenja. U domenama gdje je struktura podataka relativno jednostavna i odnosi između elemenata podataka su jednostavni, automatizirani ispravci mogu se primijeniti s visokim stupnjem pouzdanosti. Na primjer, ispravljanje problema s formatiranjem ili provođenje osnovnih ograničenja tipova podataka često mogu učinkovito obraditi radnici za samozacjeljivanje podataka.

Međutim, kako se složenost domene podataka povećava, rastu i izazovi povezani s automatiziranim ispravljanjem podataka. U domenama sa složenom poslovnom logikom, kompleksnim odnosima između entiteta podataka ili pravilima i iznimkama specifičnim za domenu, tehnike samozacjeljivanja podataka možda neće uvijek uhvatiti nijanse i mogu uvesti neželjene posljedice.

Razmotrimo primjer složene domene: sustav za trgovanje financijskim instrumentima. U ovoj domeni, podaci uključuju različite financijske instrumente, tržišne podatke, pravila trgovanja i regulatorne zahtjeve. Odnosi između različitih elemenata podataka mogu biti složeni, a pravila koja reguliraju valjanost i konzistentnost podataka mogu biti vrlo specifična za domenu.

U tako složenoj domeni, radnik za samozacjeljivanje podataka zadužen za ispravljanje nedosljednosti u podacima o trgovanju morao bi imati duboko razumijevanje pravila i ograničenja specifičnih za domenu. Morao bi uzeti u obzir faktore kao što su tržišne regulacije, ograničenja trgovanja, izračuni rizika i postupci namire. Automatizirani ispravci u ovom kontekstu možda neće uvijek obuhvatiti punu složenost domene i mogu nenamjerno uvesti pogreške ili prekršiti pravila specifična za domenu.

Za rješavanje izazova složenosti domene, tehnike samozacjeljivanja podataka mogu se poboljšati uključivanjem znanja i pravila specifičnih za domenu u AI modele i radnike. To se može postići kroz tehnike kao što su:

1. **Treniranje specifično za domenu**: AI modeli koji se koriste za samozacjeljivanje podataka mogu biti usmjereni ili čak fino podešeni na skupovima podataka specifičnim za domenu koji obuhvaćaju složenosti i pravila određene domene. Izlaganjem modela reprezentativnim podacima i scenarijima, oni mogu naučiti obrasce, ograničenja i iznimke specifične za domenu.
2. **Ograničenja temeljena na pravilima**: Radnici za samozacjeljivanje podataka mogu se nadograditi eksplicitnim ograničenjima temeljenim na pravilima koja kodiraju znanje specifično za domenu. Ova pravila mogu definirati domenski stručnjaci i integrirati ih u proces ispravljanja podataka. AI modeli tada mogu koristiti ta pravila za usmjeravanje svojih odluka i osiguravanje usklađenosti sa zahtjevima specifičnim za domenu.
3. **Suradnja s domenskim stručnjacima**: U složenim domenama ključno je uključiti domenske stručnjake u dizajn i razvoj tehnika samozacjeljivanja podataka. Domenski stručnjaci mogu pružiti vrijedne uvide u složenosti podataka, poslovna pravila i potencijalne granične slučajeve. Njihovo znanje može se ugraditi u AI modele i radnike kako bi se poboljšala točnost i pouzdanost automatiziranih ispravaka podataka koristeći obrasce čovjeka u petlji.
4. **Inkrementalni i iterativni pristup**: Pri radu sa složenim domenama, često je korisno usvojiti inkrementalni i iterativni pristup samozacjeljivanju podataka. Umjesto pokušaja automatizacije ispravaka za cijelu domenu odjednom, fokusirajte se na specifične poddomene ili kategorije podataka gdje su pravila i ograničenja dobro shvaćena. Postupno proširujte opseg tehnika samozacjeljivanja kako raste razumijevanje domene i tehnike se pokažu učinkovitima.

Uzimajući u obzir složenost podatkovne domene i ugrađivanjem znanja specifičnog za domenu u tehnike samoopravljajućih podataka, možete postići ravnotežu između

automatizacije i točnosti. Važno je prepoznati da samoopravljajući podaci nisu univerzalno rješenje te da pristup treba prilagoditi specifičnim zahtjevima i izazovima svake domene.

U složenim domenama, hibridni pristup koji kombinira tehnike samoopravljajućih podataka s ljudskom stručnošću i nadzorom može biti najučinkovitiji. Automatske korekcije mogu se nositi s rutinskim i dobro definiranim slučajevima, dok se složeni scenariji ili iznimke mogu označiti za ljudski pregled i intervenciju. Ovaj suradnički pristup osigurava da se ostvare prednosti automatizacije uz održavanje potrebne kontrole i točnosti u složenim podatkovnim domenama.

Objašnjivost i transparentnost

Objašnjivost se odnosi na sposobnost razumijevanja i interpretacije razloga iza odluka koje donose AI modeli, dok transparentnost uključuje pružanje jasnog uvida u proces ispravljanja podataka.

U mnogim kontekstima, izmjene podataka moraju biti podložne reviziji i opravdane. Dionici, uključujući poslovne korisnike, revizore i regulatorna tijela, mogu zahtijevati objašnjenja zašto su određene korekcije podataka napravljene i kako su AI modeli došli do tih odluka. Ovo je posebno ključno u domenama gdje točnost i integritet podataka imaju značajne implikacije, kao što su financije, zdravstvo i pravna pitanja.

Kako bi se odgovorilo na potrebu za objašnjivošću i transparentnošću, tehnike samoopravljajućih podataka trebaju uključivati mehanizme koji pružaju uvid u proces donošenja odluka AI modela. To se može postići kroz različite pristupe:

1. **Lanac razmišljanja**: Traženje od modela da objasni svoje razmišljanje “naglas” prije primjene promjena na podacima može omogućiti lakše razumijevanje procesa donošenja odluka i može generirati objašnjenja razumljiva ljudima za napravljene korekcije. Kompromis je malo veća složenost u odvajanju objašnjenja od strukturiranog izlaza podataka, što se može riješiti…

2. **Generiranje objašnjenja**: Sustavi za samoopravljanje podataka mogu biti opremljeni sposobnošću generiranja objašnjenja razumljivih ljudima za korekcije koje provode. To se može postići tako da se od modela traži da prikaže svoj proces donošenja odluka kao lako razumljiva objašnjenja *integrirana u same podatke*. Na primjer, sustav za samoopravljanje podataka mogao bi generirati izvješće koje ističe specifične nedosljednosti u podacima koje je identificirao, korekcije koje je primijenio i obrazloženje tih korekcija.
3. **Važnost značajki**: AI modeli mogu biti upućeni s informacijama o važnosti različitih značajki ili atributa u procesu ispravljanja podataka kao dio njihovih uputa. Te upute se zatim mogu pokazati ljudskim dionicima. Identificiranjem ključnih faktora koji utječu na odluke modela, dionici mogu steći uvid u obrazloženje iza korekcija i procijeniti njihovu valjanost.
4. **Bilježenje i revizija**: Implementacija sveobuhvatnih mehanizama bilježenja i revizije ključna je za održavanje transparentnosti u procesu samoopravljanja podataka. Svaka korekcija podataka koju naprave AI modeli treba biti zabilježena, uključujući originalne podatke, ispravljene podatke i specifične poduzete radnje. Ovaj revizijski trag omogućuje retrospektivnu analizu i pruža jasan zapis o izmjenama napravljenim na podacima.
5. **Pristup s ljudskom kontrolom**: Uključivanje pristupa s ljudskom kontrolom može poboljšati objašnjivost i transparentnost tehnika samoopravljajućih podataka. Uključivanjem ljudskih stručnjaka u pregled i validaciju korekcija generiranih AI-jem, organizacije mogu osigurati da su korekcije usklađene s domenskim znanjem i poslovnim zahtjevima. Ljudski nadzor dodaje dodatni sloj odgovornosti i omogućuje identifikaciju potencijalnih pristranosti ili grešaka u AI modelima.
6. **Kontinuirano praćenje i evaluacija**: Redovito praćenje i evaluacija performansi tehnika samoopravljajućih podataka ključni su za održavanje transparentnosti i povjerenja. Procjenom točnosti i učinkovitosti AI modela tijekom vremena, organizacije mogu identificirati odstupanja ili anomalije i poduzeti korektivne mjere. Kontinuirano praćenje pomaže osigurati da proces samoopravljanja

podataka ostane pouzdan i usklađen sa željenim ishodima.

Objašnjivost i transparentnost su ključni aspekti pri implementaciji tehnika samoopravljajućih podataka. Pružanjem jasnih objašnjenja za korekcije podataka, održavanjem sveobuhvatnih revizijskih tragova i uključivanjem ljudskog nadzora, organizacije mogu izgraditi povjerenje u proces samoopravljanja podataka i osigurati da su izmjene napravljene na podacima opravdane i usklađene s poslovnim ciljevima.

Važno je postići ravnotežu između prednosti automatizacije i potrebe za transparentnošću. Iako tehnike samoopravljajućih podataka mogu značajno poboljšati kvalitetu podataka i učinkovitost, to ne bi trebalo biti na štetu gubitka vidljivosti i kontrole nad procesom ispravljanja podataka. Dizajniranjem sustava za samoopravljanje podataka s fokusom na objašnjivost i transparentnost, organizacije mogu iskoristiti snagu AI-ja uz održavanje potrebne razine odgovornosti i povjerenja u podatke.

Nenamjerne posljedice

Iako tehnike samoopravljajućih podataka imaju za cilj poboljšati kvalitetu i konzistentnost podataka, ključno je biti svjestan potencijalnih nenamjernih posljedica. Automatske korekcije, ako nisu pažljivo dizajnirane i nadzirane, mogu nehotice izmijeniti značenje ili kontekst podataka, dovodeći do problema u kasnijim fazama.

Jedan od primarnih rizika samoopravljajućih podataka je uvođenje pristranosti ili grešaka u proces ispravljanja podataka. AI modeli, kao i bilo koji drugi softverski sustav, mogu biti podložni pristranostima prisutnim u podacima za treniranje ili uvedenim kroz dizajn algoritama. Ako se te pristranosti ne identificiraju i ne ublaže, mogu se proširiti kroz proces samoopravljanja podataka i rezultirati iskrivljenim ili netočnim izmjenama podataka.

Na primjer, razmotrimo samoiscjeljujućeg podatkovnog radnika zaduženog za ispravljanje nedosljednosti u demografskim podacima kupaca. Ako je AI model

naučio pristranosti iz povijesnih podataka, poput povezivanja određenih zanimanja ili razina prihoda s određenim spolovima ili etničkim skupinama, može donijeti pogrešne pretpostavke i modificirati podatke na način koji pojačava te pristranosti. To može dovesti do netočnih profila kupaca, pogrešnih poslovnih odluka i potencijalno diskriminirajućih ishoda.

Još jedna potencijalna neželjena posljedica je gubitak vrijednih informacija ili konteksta tijekom procesa ispravljanja podataka. Tehnike samoiscjeljivanja podataka često se usredotočuju na standardizaciju i normalizaciju podataka kako bi se osigurala dosljednost. Međutim, u nekim slučajevima, izvorni podaci mogu sadržavati nijanse, iznimke ili kontekstualne informacije koje su važne za razumijevanje cjelovite slike. Automatska ispravljanja koja slijepo provode standardizaciju mogu nenamjerno ukloniti ili zamagliti te vrijedne informacije.

Na primjer, zamislite samoiscjeljujućeg podatkovnog radnika odgovornog za ispravljanje nedosljednosti u medicinskoj dokumentaciji. Ako radnik naiđe na medicinsku povijest pacijenta s rijetkim stanjem ili neuobičajenim planom liječenja, mogao bi pokušati normalizirati podatke kako bi odgovarali uobičajenijem obrascu. Međutim, čineći to, mogao bi izgubiti specifične detalje i kontekst koji su ključni za točno predstavljanje jedinstvene situacije pacijenta. Ovaj gubitak informacija može imati ozbiljne posljedice za skrb o pacijentu i donošenje medicinskih odluka.

Za ublažavanje rizika od neželjenih posljedica, ključno je zauzeti proaktivan pristup pri dizajniranju i implementaciji tehnika samoiscjeljivanja podataka:

1. **Temeljito testiranje i validacija**: Prije implementacije samoiscjeljujućih podatkovnih radnika u produkciju, ključno je temeljito testirati i validirati njihovo ponašanje u različitim scenarijima. To uključuje testiranje s reprezentativnim skupovima podataka koji pokrivaju različite granične slučajeve, iznimke i potencijalne pristranosti. Rigorozno testiranje pomaže identificirati i riješiti sve neželjene posljedice prije nego što utječu na podatke u stvarnom svijetu.

2. **Kontinuirano praćenje i evaluacija**: Implementacija mehanizama za kontinuirano praćenje i evaluaciju ključna je za otkrivanje i ublažavanje neželjenih posljedica tijekom vremena. Redovito pregledavanje ishoda procesa samoiscjeljivanja podataka, analiza utjecaja na povezane sustave i donošenje odluka, te prikupljanje povratnih informacija od dionika može pomoći u identificiranju štetnih učinaka i potaknuti pravovremene korektivne akcije. Ako vaša organizacija ima operativne nadzorne ploče, vjerojatno je dobra ideja dodati jasno vidljive metrike povezane s automatiziranim promjenama podataka. Dodavanje alarma povezanih s velikim odstupanjima od normalne aktivnosti promjene podataka vjerojatno je još bolja ideja!
3. **Ljudski nadzor i intervencija**: Održavanje ljudskog nadzora i mogućnosti intervencije u procesu samoiscjeljivanja podataka je ključno. Iako automatizacija može značajno poboljšati učinkovitost, važno je da ljudski stručnjaci pregledavaju i validiraju ispravke koje rade AI modeli, posebno u kritičnim ili osjetljivim domenama. Ljudska prosudba i stručnost u domeni mogu pomoći u identificiranju i rješavanju svih neželjenih posljedica koje se mogu pojaviti.

4. **Objašnjivi UI (XAI) i transparentnost**: Kao što je raspravljeno u prethodnom pododjeljku, uključivanje tehnika objašnjivog UI-ja i osiguravanje transparentnosti u procesu samoiscjeljivanja podataka može pomoći u ublažavanju neželjenih posljedica. Pružanjem jasnih objašnjenja za ispravke podataka i održavanjem sveobuhvatnih revizijskih zapisa, organizacije mogu bolje razumjeti i pratiti obrazloženje iza modifikacija koje rade AI modeli.
5. **Inkrementalni i iterativni pristup**: Usvajanje inkrementalnog i iterativnog pristupa samoiscjeljivanju podataka može pomoći u minimiziranju rizika od neželjenih posljedica. Umjesto primjene automatskih ispravaka na cijeli skup podataka odjednom, počnite s podskupom podataka i postupno proširujte opseg kako se tehnike pokažu učinkovitima i pouzdanima. To omogućuje pažljivo praćenje i prilagodbu tijekom procesa, smanjujući utjecaj bilo kakvih neželjenih posljedica.

6. **Suradnja i povratne informacije**: Uključivanje dionika iz različitih domena i poticanje suradnje i povratnih informacija tijekom procesa samoiscjeljivanja podataka može pomoći u identificiranju i rješavanju neželjenih posljedica. Redovito traženje inputa od stručnjaka u domeni, korisnika podataka i krajnjih korisnika može pružiti vrijedne uvide u stvarni utjecaj ispravaka podataka i istaknuti probleme koji su možda bili previđeni.

Proaktivnim rješavanjem rizika od neželjenih posljedica i implementacijom odgovarajućih zaštitnih mjera, organizacije mogu iskoristiti prednosti tehnika samoiscjeljivanja podataka uz minimiziranje potencijalnih štetnih učinaka. Važno je pristupiti samoiscjeljivanju podataka 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.

Pri razmatranju korištenja obrazaca samoiscjeljivanja podataka, ključno je pažljivo evaluirati ove faktore i odvagnuti prednosti u odnosu na potencijalne rizike i ograničenja. U nekim slučajevima, hibridni pristup koji kombinira automatske ispravke s ljudskim nadzorom i intervencijom može biti najprikladnije rješenje.

Također je vrijedno napomenuti da tehnike samoiscjeljivanja podataka ne bi trebale biti shvaćene kao zamjena za robusnu validaciju podataka, sanitizaciju unosa i mehanizme obrade pogrešaka. Ove temeljne prakse ostaju kritične za osiguravanje integriteta i sigurnosti podataka. Samoiscjeljivanje podataka treba promatrati kao komplementarni pristup koji može nadopuniti i poboljšati ove postojeće mjere.

U konačnici, odluka o korištenju obrazaca samoiscjeljivanja podataka ovisi o specifičnim zahtjevima, ograničenjima i prioritetima vaše aplikacije. Pažljivim razmatranjem gore navedenih faktora i njihovim usklađivanjem s ciljevima i arhitekturom vaše aplikacije, možete donijeti informirane odluke o tome kada i kako učinkovito iskoristiti tehnike samoiscjeljivanja podataka.

Kontekstualno generiranje sadržaja

Obrasci kontekstualnog generiranja sadržaja koriste snagu velikih jezičnih modela (LLM) za generiranje dinamičkog i kontekstualno specifičnog sadržaja unutar aplikacija. Ova kategorija obrazaca prepoznaje važnost isporuke personaliziranog i relevantnog sadržaja korisnicima na temelju njihovih specifičnih potreba, preferencija, pa čak i prethodnih i trenutnih interakcija s aplikacijom.

> U kontekstu ovog pristupa, "sadržaj" se odnosi i na primarni sadržaj (tj. blog objave, članke itd.) i na meta-sadržaj, poput preporuka za primarni sadržaj.

Obrasci kontekstualnog generiranja sadržaja mogu igrati ključnu ulogu u poboljšanju razine angažmana korisnika, pružanju prilagođenih iskustava i automatizaciji zadataka stvaranja sadržaja kako za vas tako i za vaše korisnike. Korištenjem obrazaca koje opisujemo u ovom poglavlju, možete stvoriti aplikacije koje dinamički generiraju sadržaj, prilagođavajući se kontekstu i ulaznim podacima u stvarnom vremenu.

Obrasci funkcioniraju integrirajući LLM-ove u izlaze aplikacije, od korisničkog sučelja (ponekad nazivanog "chrome"), do e-pošte i drugih oblika obavijesti, kao i bilo kojih procesa generiranja sadržaja.

Kada korisnik komunicira s aplikacijom ili pokrene specifični zahtjev za sadržajem, aplikacija bilježi relevantni kontekst, poput korisničkih preferencija, prethodnih interakcija ili specifičnih upita. Te kontekstualne informacije se zatim unose u LLM, zajedno sa svim potrebnim predlošcima ili smjernicama i koriste se za proizvodnju tekstualnog izlaza koji bi inače morao biti ili hardkodiran, pohranjen u bazi podataka ili algoritmički generiran.

Sadržaj generiran LLM-om može poprimiti različite oblike, kao što su personalizirane preporuke, dinamički opisi proizvoda, prilagođeni odgovori e-poštom, pa čak i cijeli članci ili blog objave. Jedna od najradikalnijih primjena ovog sadržaja koju sam započeo prije više od godinu dana je dinamičko generiranje elemenata korisničkog sučelja poput oznaka obrazaca, opisa alata i drugih vrsta objašnjavajućeg teksta.

Personalizacija

Jedna od ključnih prednosti obrazaca kontekstualnog generiranja sadržaja je mogućnost pružanja visoko personaliziranih iskustava korisnicima. Generiranjem sadržaja na temelju konteksta specifičnog za korisnika, ovi obrasci omogućuju aplikacijama da prilagode sadržaj individualnim interesima, preferencijama i interakcijama korisnika.

Personalizacija nadilazi jednostavno umetanje korisničkog imena u generički sadržaj. Uključuje korištenje bogatog konteksta dostupnog o svakom korisniku za generiranje

sadržaja koji rezonira s njihovim specifičnim potrebama i željama. Taj kontekst može uključivati širok raspon čimbenika, kao što su:

1. **Podaci korisničkog profila**: Na najopćenitijoj razini primjene ove tehnike, demografski podaci, interesi, preferencije i drugi atributi profila mogu se koristiti za generiranje sadržaja koji se usklađuje s korisnikovom pozadinom i karakteristikama.
2. **Podaci o ponašanju**: Prethodne interakcije korisnika s aplikacijom, poput pregledanih stranica, kliknutih poveznica ili kupljenih proizvoda, mogu pružiti vrijedne uvide u njihovo ponašanje i interese. Ti se podaci mogu koristiti za generiranje prijedloga sadržaja koji odražava njihove obrasce angažmana i predviđa njihove buduće potrebe.
3. **Kontekstualni čimbenici**: Trenutni kontekst korisnika, poput njihove lokacije, uređaja, doba dana, pa čak i vremenske prognoze, može utjecati na proces generiranja sadržaja. Na primjer, aplikacija za putovanja mogla bi imati AI radnika koji može generirati personalizirane preporuke na temelju trenutne lokacije korisnika i prevladavajućih vremenskih uvjeta.

Korištenjem ovih kontekstualnih čimbenika, obrasci kontekstualnog generiranja sadržaja omogućuju aplikacijama isporuku sadržaja koji djeluje kao da je posebno izrađen za svakog pojedinog korisnika. Ova razina personalizacije ima nekoliko značajnih prednosti:

1. **Povećani angažman**: Personalizirani sadržaj privlači pažnju korisnika i održava ih angažiranima u aplikaciji. Kada korisnici osjećaju da je sadržaj relevantan i da izravno odgovara njihovim potrebama, vjerojatnije je da će provesti više vremena u interakciji s aplikacijom i istraživanju njezinih značajki.
2. **Poboljšano zadovoljstvo korisnika**: Personalizirani sadržaj pokazuje da aplikacija razumije i brine o jedinstvenim zahtjevima korisnika. Pružanjem sadržaja koji je koristan, informativan i usklađen s njihovim interesima,

aplikacija može poboljšati zadovoljstvo korisnika i izgraditi snažniju vezu sa svojim korisnicima.

3. **Više stope konverzije**: U kontekstu e-trgovine ili marketinških aplikacija, personalizirani sadržaj može značajno utjecati na stope konverzije. Predstavljanjem proizvoda, ponuda ili preporuka koje su prilagođene njihovim preferencijama i ponašanju, aplikacija može povećati vjerojatnost da će korisnici poduzeti željene radnje, poput kupnje ili prijave za uslugu.

Produktivnost

Obrasci kontekstualnog generiranja sadržaja mogu značajno povećati određene vrste produktivnosti smanjujući potrebu za ručnim generiranjem i uređivanjem sadržaja u kreativnim procesima. Korištenjem snage LLM-ova, možete generirati kvalitetan sadržaj u većem opsegu, štedeći vrijeme i trud koji bi vaši kreatori sadržaja i razvojni programeri inače morali utrošiti na zamoran ručni rad.

Tradicionalno, stvaratelji sadržaja moraju istraživati, pisati, uređivati i formatirati sadržaj kako bi osigurali da zadovoljava zahtjeve aplikacije i očekivanja korisnika. Ovaj proces može biti vremenski zahtjevan i resursno intenzivan, posebno kako raste količina sadržaja.

Međutim, s obrascima kontekstualnog generiranja sadržaja, proces stvaranja sadržaja može biti uvelike automatiziran. LLM-ovi mogu generirati koherentan, gramatički ispravan i kontekstualno relevantan sadržaj na temelju zadanih uputa i smjernica. Ova automatizacija nudi nekoliko prednosti za produktivnost:

1. **Smanjeni ručni rad**: Delegiranjem zadataka generiranja sadržaja LLM-ovima, stvaratelji sadržaja mogu se usredotočiti na zadatke više razine kao što su strategija sadržaja, osmišljavanje ideja i osiguranje kvalitete. Mogu pružiti potreban kontekst, predloške i smjernice LLM-u i pustiti ga da se bavi stvarnim

generiranjem sadržaja. To smanjuje ručni napor potreban za pisanje i uređivanje, omogućujući stvarateljima sadržaja da budu produktivniji i učinkovitiji.

2. **Brže stvaranje sadržaja**: LLM-ovi mogu generirati sadržaj mnogo brže od ljudskih pisaca. S pravim uputama i smjernicama, LLM može proizvesti više dijelova sadržaja u nekoliko sekundi ili minuta. Ova brzina omogućuje aplikacijama da generiraju sadržaj mnogo bržim tempom, držeći korak sa zahtjevima korisnika i stalno promjenjivim digitalnim okruženjem.

> Vodi li brže stvaranje sadržaja do situacije "tragedije zajedničkog dobra" gdje se internet guši u sadržaju koji nitko ne čita? Nažalost, sumnjam da je odgovor da.

3. **Konzistentnost i kvaliteta**: LLM-ovi mogu bez problema revidirati sadržaj tako da bude konzistentan u stilu, tonu i kvaliteti. Uz jasne smjernice i primjere, određene vrste aplikacija (npr. redakcije, odnosi s javnošću itd.) mogu osigurati da njihov sadržaj koji stvaraju ljudi bude usklađen s glasom brenda i zadovoljava željene standarde kvalitete. Ova konzistentnost smanjuje potrebu za opsežnim uređivanjem i revizijama, štedeći vrijeme i trud u procesu stvaranja sadržaja.
4. **Iteracija i optimizacija:** Obrasci kontekstualnog generiranja sadržaja omogućuju brzu iteraciju i optimizaciju sadržaja. Prilagođavanjem uputa, predložaka ili smjernica pruženih LLM-u, vaše aplikacije mogu brzo generirati varijacije sadržaja i testirati različite pristupe na automatizirani način koji nikada prije nije bio moguć. Ovaj iterativni proces omogućuje brže eksperimentiranje i usavršavanje strategija sadržaja, što s vremenom dovodi do učinkovitijeg i angažiranijeg sadržaja. Ova konkretna tehnika može biti potpuna prekretnica za aplikacije poput e-trgovine koje žive i umiru na temelju stopa napuštanja i angažmana

Važno je napomenuti da iako obrasci kontekstualnog generiranja sadržaja mogu značajno poboljšati produktivnost, oni ne eliminiraju potpuno potrebu za ljudskim sudjelovanjem. Stvaratelji sadržaja i urednici i dalje imaju ključnu ulogu u definiranju cjelokupne strategije sadržaja, pružanju smjernica LLM-u i osiguravanju kvalitete i prikladnosti generiranog sadržaja.

Automatizacijom repetitivnijih i vremenski zahtjevnijih aspekata stvaranja sadržaja, obrasci kontekstualnog generiranja sadržaja oslobađaju dragocjeno ljudsko vrijeme i resurse koji se mogu preusmjeriti na zadatke više vrijednosti. Ova povećana produktivnost omogućuje vam da isporučite personaliziraniji i angažiraniji sadržaj korisnicima uz optimizaciju radnih tokova stvaranja sadržaja.

Brza iteracija i eksperimentiranje

Obrasci kontekstualnog generiranja sadržaja omogućuju vam brzu iteraciju i eksperimentiranje s različitim varijacijama sadržaja, omogućujući bržu optimizaciju i usavršavanje vaše strategije sadržaja. Možete generirati više verzija sadržaja u nekoliko sekundi, jednostavno prilagođavanjem konteksta, predložaka ili smjernica pruženih modelu.

Ova mogućnost brze iteracije nudi nekoliko ključnih prednosti:

1. **Testiranje i optimizacija**: S mogućnošću brzog generiranja varijacija sadržaja, možete lako testirati različite pristupe i mjeriti njihovu učinkovitost. Na primjer, možete generirati više verzija opisa proizvoda ili marketinške poruke, svaku prilagođenu određenom segmentu korisnika ili kontekstu. Analiziranjem metrika angažmana korisnika, poput stopa klikova ili konverzija, možete identificirati najučinkovitije varijacije sadržaja i u skladu s tim optimizirati svoju strategiju sadržaja.

2. **A/B testiranje**: Obrasci kontekstualnog generiranja sadržaja omogućuju besprijekorno A/B testiranje sadržaja. Možete generirati dvije ili više varijacija sadržaja i nasumično ih servirati različitim grupama korisnika. Uspoređujući performanse svake varijacije, možete odrediti koji sadržaj najbolje rezonira s vašom ciljnom publikom. Ovaj pristup temeljen na podacima omogućuje vam donošenje informiranih odluka i kontinuirano usavršavanje sadržaja kako biste maksimizirali angažman korisnika i postigli željene rezultate.
3. **Eksperimenti s personalizacijom**: Brza iteracija i eksperimentiranje posebno su vrijedni kada je riječ o personalizaciji. S obrascima kontekstualnog generiranja sadržaja, možete brzo generirati personalizirane varijacije sadržaja temeljene na različitim segmentima korisnika, preferencijama ili ponašanjima. Eksperimentiranjem s različitim strategijama personalizacije, možete identificirati najučinkovitije pristupe za angažiranje pojedinačnih korisnika i pružanje prilagođenih iskustava.
4. **Prilagodba promjenjivim trendovima**: Mogućnost brze iteracije i eksperimentiranja omogućuje vam da ostanete agilni i prilagodite se promjenjivim trendovima i korisničkim preferencijama. Kako se pojavljuju nove teme, ključne riječi ili korisnička ponašanja, možete brzo generirati sadržaj koji je usklađen s tim trendovima. Kontinuiranim eksperimentiranjem i usavršavanjem sadržaja možete ostati relevantni i održati konkurentsku prednost u stalno razvijajućem digitalnom okruženju.
5. **Ekonomično eksperimentiranje**: Tradicionalno eksperimentiranje sa sadržajem često uključuje značajno vrijeme i resurse, jer stvaratelji sadržaja trebaju ručno razvijati i testirati različite varijacije. Međutim, s obrascima kontekstualnog generiranja sadržaja, trošak eksperimentiranja značajno je smanjen. LLM-ovi mogu brzo generirati varijacije sadržaja u većem opsegu, omogućujući vam da istražite širok raspon ideja i pristupa bez značajnih troškova.

Za maksimalno iskorištavanje brze iteracije i eksperimentiranja, važno je imati dobro definiran eksperimentalni okvir. Taj okvir trebao bi uključivati:

- Jasne ciljeve i hipoteze za svaki eksperiment
- Odgovarajuće metrike i mehanizme praćenja za mjerenje učinkovitosti sadržaja
- Strategije segmentacije i ciljanja kako bi se osiguralo da odgovarajuće varijacije sadržaja dođu do pravih korisnika
- Alate za analizu i izvještavanje za izvlačenje uvida iz eksperimentalnih podataka
- Proces za ugrađivanje naučenog i optimizacija u vašu strategiju sadržaja

Prihvaćanjem brze iteracije i eksperimentiranja možete kontinuirano usavršavati i optimizirati svoj sadržaj, osiguravajući da ostane privlačan, relevantan i učinkovit u postizanju ciljeva vaše aplikacije. Ovaj agilni pristup stvaranju sadržaja omogućuje vam da budete korak ispred i pružite izvanredno korisničko iskustvo.

Skalabilnost i učinkovitost

Kako aplikacije rastu i povećava se potražnja za personaliziranim sadržajem, obrasci kontekstualnog generiranja sadržaja omogućuju učinkovito skaliranje stvaranja sadržaja. LLM-ovi mogu istovremeno generirati sadržaj za velik broj korisnika i konteksta, bez potrebe za proporcionalnim povećanjem ljudskih resursa. Ova skalabilnost omogućuje aplikacijama da pruže personalizirano iskustvo rastućem broju korisnika bez opterećenja njihovih mogućnosti stvaranja sadržaja.

Imajte na umu da se kontekstualno generiranje sadržaja može učinkovito koristiti za internacionalizaciju vaše aplikacije “u hodu”. Zapravo, upravo sam to učinio koristeći svoj Instant18n Gem za isporuku Olympije na više od pola desetaka jezika, iako postojimo manje od godinu dana.

AI pogonjena lokalizacija

Ako mi dopustite da se na trenutak pohvalim, mislim da je moja Instant18n biblioteka za Rails aplikacije revolucionaran primjer obrasca “Kontekstualnog generiranja sadržaja” u

akciji, pokazujući transformativni potencijal AI-ja u razvoju aplikacija. Ovaj gem koristi snagu OpenAI-jevog GPT velikog jezičnog modela kako bi revolucionirao način na koji se upravlja internacionalizacijom i lokalizacijom u Rails aplikacijama.

Tradicionalno, internacionalizacija Rails aplikacije uključuje ručno definiranje ključeva za prijevod i pružanje odgovarajućih prijevoda za svaki podržani jezik. Ovaj proces može biti dugotrajan, resursno zahtjevan i sklon nedosljednostima. Međutim, s Instant18n gemom, paradigma lokalizacije potpuno je redefinirana.

Integracijom velikog jezičnog modela, Instant18n gem omogućuje generiranje prijevoda u realnom vremenu, temeljeno na kontekstu i značenju teksta. Umjesto oslanjanja na unaprijed definirane ključeve za prijevod i statičke prijevode, gem dinamički prevodi tekst koristeći snagu AI-ja. Ovaj pristup nudi nekoliko ključnih prednosti:

1. **Besprijekorna lokalizacija**: S Instant18n gemom, programeri više ne trebaju ručno definirati i održavati datoteke prijevoda za svaki podržani jezik. Gem automatski generira prijevode na temelju pruženog teksta i željenog ciljnog jezika, čineći proces lokalizacije jednostavnim i besprijekornim.
2. **Kontekstualna točnost**: AI-ju se može dati dovoljno konteksta za razumijevanje nijansa teksta koji se prevodi. Može uzeti u obzir okolni kontekst, idiome i kulturološke reference kako bi generirao prijevode koji su točni, prirodni i kontekstualno prikladni.
3. **Opsežna jezična podrška**: Instant18n gem koristi ogromno znanje i jezične sposobnosti GPT-a, omogućujući prijevode na širok raspon jezika. Od uobičajenih jezika poput španjolskog i francuskog do opskurnijih ili izmišljenih jezika poput klingonskog i vilenskog, gem može obraditi širok spektar prevoditeljskih zahtjeva.
4. **Fleksibilnost i kreativnost**: Gem nadilazi tradicionalne jezične prijevode i omogućuje 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 predmemoriranja za poboljšanje performansi i smanjenje opterećenja kod ponovljenih prijevoda. Prevedeni tekst se predmemorira, omogućujući da se naknadni zahtjevi za istim prijevodom brzo posluže bez potrebe za redundantnim API pozivima.

Instant18n gem primjer je snage obrasca "Kontekstualnog generiranja sadržaja" korištenjem AI-ja za dinamičko generiranje lokaliziranog sadržaja. Pokazuje kako se AI može integrirati u osnovnu funkcionalnost Rails aplikacije, transformirajući način na koji programeri pristupaju internacionalizaciji i lokalizaciji.

Eliminiranjem potrebe za ručnim upravljanjem prijevodima i omogućavanjem prijevoda u realnom vremenu temeljenih na kontekstu, Instant18n gem štedi programerima značajno vrijeme i trud. Omogućuje im da se usredotoče na izgradnju osnovnih značajki svoje aplikacije, istovremeno osiguravajući da se aspekt lokalizacije odvija besprijekorno i precizno.

Važnost korisničkog testiranja i povratnih informacija

Na kraju, uvijek imajte na umu važnost korisničkog testiranja i povratnih informacija. Ključno je potvrditi da kontekstualno generiranje sadržaja zadovoljava očekivanja korisnika i usklađeno je s ciljevima aplikacije. Kontinuirano poboljšavajte i usavršavajte generirani sadržaj na temelju korisničkih uvida i analitike. Ako generirate dinamički sadržaj u velikom opsegu koji bi bilo nemoguće ručno validirati od strane vas i vašeg tima, razmislite o dodavanju mehanizama za povratne informacije koji omogućuju korisnicima prijavu čudnog ili pogrešnog sadržaja, zajedno s objašnjenjem zašto. Te dragocjene povratne informacije mogu se čak proslijediti AI sustavu zaduženom za prilagodbu komponente koja je generirala sadržaj!

Generativno korisničko sučelje

Pažnja je danas toliko dragocjena da učinkovito korisničko angažiranje zahtijeva softverska iskustva koja nisu samo besprijekorna i intuitivna, već i visoko personalizirana prema individualnim potrebama, preferencijama i kontekstima. Zbog toga se dizajneri i programeri sve češće suočavaju s izazovom stvaranja korisničkih sučelja koja se mogu prilagoditi i udovoljiti jedinstvenim zahtjevima svakog korisnika *u velikim razmjerima.*

Generativno korisničko sučelje (GenUI) je istinski revolucionaran pristup dizajnu korisničkog sučelja koji koristi snagu velikih jezičnih modela (LLM) za stvaranje visoko personaliziranih i dinamičnih korisničkih iskustava u stvarnom vremenu. Želio sam vam u ovoj knjizi barem dati uvod u GenUI jer vjerujem da je to jedna od najotvorenijih prilika koje trenutno postoje u području dizajna aplikacija i razvojnih okvira. Uvjeren sam da će se u ovoj specifičnoj niši pojaviti desetci ili više novih uspješnih komercijalnih i projekata otvorenog koda.

U svojoj srži, GenUI kombinira principe Kontekstualnog generiranja sadržaja s naprednim AI tehnikama za dinamičko generiranje elemenata korisničkog sučelja, poput teksta, slika i rasporeda, temeljeno na dubokom razumijevanju korisničkog konteksta, preferencija i ciljeva. GenUI omogućuje dizajnerima i programerima stvaranje sučelja koja se prilagođavaju i razvijaju kao odgovor na korisničke interakcije, pružajući razinu personalizacije koja je prije bila nedostižna.

GenUI predstavlja temeljnu promjenu u načinu na koji pristupamo dizajnu korisničkog sučelja. Umjesto dizajniranja za mase, GenUI nam omogućuje dizajniranje za pojedinca. Personalizirani sadržaj i sučelja imaju potencijal stvaranja korisničkih iskustava koja rezoniraju sa svakim korisnikom na dubljoj razini, povećavajući angažman, zadovoljstvo i lojalnost.

Kao najnovija tehnika, prelazak na GenUI pun je konceptualnih i praktičnih izazova. Integracija AI-ja u proces dizajna, osiguravanje da generirana sučelja nisu samo personalizirana već i upotrebljiva, pristupačna i usklađena s ukupnim brendom i korisničkim iskustvom, sve su to izazovi koji čine GenUI područjem za malobrojne, ne za mnoge. Dodatno, uključivanje AI-ja postavlja pitanja o privatnosti podataka, transparentnosti, pa čak i etičkim implikacijama.

Unatoč izazovima, personalizirana iskustva u velikim razmjerima imaju moć potpuno transformirati način na koji komuniciramo s digitalnim proizvodima i uslugama. To otvara mogućnosti za stvaranje inkluzivnih i pristupačnih sučelja koja zadovoljavaju raznolike potrebe korisnika, bez obzira na njihove sposobnosti, pozadinu ili preferencije.

U ovom poglavlju istražit ćemo koncept GenUI-ja, proučavajući neke definirajuće karakteristike, ključne prednosti i potencijalne izazove. Počinjemo razmatranjem najosnovnijeg i najpristupačnijeg oblika GenUI-ja: generiranje tekstualnog sadržaja za inače tradicionalno dizajnirana i implementirana korisnička sučelja.

Generiranje teksta za korisnička sučelja

Tekstualni elementi koji postoje u sučelju vaše aplikacije, poput oznaka obrazaca, opisa alata i objašnjenja, obično su fiksno kodirani u predloške ili UI komponente, pružajući dosljedno ali generičko iskustvo za sve korisnike. Koristeći obrasce kontekstualnog generiranja sadržaja, možete transformirati ove statične elemente u dinamične, kontekstualno svjesne i personalizirane komponente.

Personalizirani obrasci

Obrasci su sveprisutni dio web i mobilnih aplikacija, služeći kao primarno sredstvo prikupljanja korisničkog unosa. Međutim, tradicionalni obrasci često predstavljaju generičko i bezlično iskustvo, sa standardnim oznakama i poljima koja se ne moraju uvijek podudarati s korisnikovim specifičnim kontekstom ili potrebama. Korisnici će vjerojatnije ispuniti obrasce koji se čine prilagođeni njihovim potrebama i preferencijama, što dovodi do većih stopa konverzije i zadovoljstva korisnika.

Međutim, važno je postići ravnotežu između personalizacije i dosljednosti. Iako prilagođavanje obrazaca pojedinačnim korisnicima može biti korisno, ključno je održati određenu razinu poznatosti i predvidljivosti. Korisnici bi i dalje trebali moći lako prepoznati i navigirati obrascima, čak i s personaliziranim elementima.

Evo nekoliko ideja za personalizirane obrasce kao inspiraciju:

Kontekstualni prijedlozi polja

GenUI može analizirati prethodne interakcije korisnika, preferencije i podatke kako bi pružio inteligentne prijedloge polja kao predviđanja. Na primjer, ako je korisnik prethodno unio svoju adresu za dostavu, obrazac može automatski popuniti relevantna polja njihovim spremljenim informacijama. Ovo ne samo da štedi vrijeme, već i pokazuje da aplikacija razumije i pamti korisničke preferencije.

Čekajte malo, nije li ova tehnika nešto što bi se moglo napraviti bez korištenja UI? Naravno, ali ljepota pokretanja ovakve funkcionalnosti pomoću umjetne inteligencije je dvojaka: 1) koliko ju je lako implementirati i 2) koliko je otporna dok se vaše korisničko sučelje mijenja i razvija tijekom vremena.

Hajdemo na brzinu složiti servis za naš teoretski sustav obrade narudžbi, koji će pokušati proaktivno popuniti odgovarajuću adresu za dostavu korisniku.

```
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
```

```
33
34    def order_history
35      customer.orders.successful.limit(100).map do |order|
36        {
37          date: order.date,
38          description: order.description,
39          shipping_address: order.shipping_address
40        }
41      end
42    end
43  end
```

Ovaj primjer je vrlo pojednostavljen, ali bi trebao funkcionirati u većini slučajeva. Ideja je pustiti umjetnu inteligenciju da pogađa na isti način kao što bi to učinio čovjek. Kako bih pojasnio o čemu govorim, pogledajmo neke uzorke podataka:

```
1  Order History:
2  [
3    {"date": "2024-01-03", "description": "garden soil mix",
4     "shipping_address": "123 Country Lane, Rural Town"},
5    {"date": "2024-01-15", "description": "hardcover fiction novels",
6     "shipping_address": "456 City Apt, Metroville"},
7    {"date": "2024-01-22", "description": "baby diapers", "shipping_address":
8     "789 Suburb St, Quietville"},
9    {"date": "2024-02-01", "description": "organic vegetables",
10    "shipping_address": "123 Country Lane, Rural Town"},
11   {"date": "2024-02-17", "description": "mystery thriller book set",
12    "shipping_address": "456 City Apt, Metroville"},
13   {"date": "2024-02-25", "description": "baby wipes",
14    "shipping_address": "789 Suburb St, Quietville"},
15   {"date": "2024-03-05", "description": "flower seeds",
16    "shipping_address": "123 Country Lane, Rural Town"},
17   {"date": "2024-03-20", "description": "biographies",
18    "shipping_address": "456 City Apt, Metroville"},
19   {"date": "2024-03-30", "description": "baby formula",
20    "shipping_address": "789 Suburb St, Quietville"},
21   {"date": "2024-04-12", "description": "lawn fertilizer",
22    "shipping_address": "123 Country Lane, Rural Town"},
23   {"date": "2024-04-22", "description": "science fiction novels",
24    "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"}
]
```

Jeste li primijetili uzorak u podacima? Garantiram vam da je ovo jednostavno za VJM. Kako bismo to pokazali, upitajmo GPT-4 koja je najvjerojatnija adresa za dostavu za "toplomjer".

```
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 pretjerivanje koristiti skupi model poput GPT-4 za ovaj zadatak, u pravu ste! Isprobao sam isti prompt na Mistral 7B Instructu i proizveo je sljedeći odgovor brzinom od 75 tokena u sekundi, uz zanemariv trošak od 0,000218 USD.

```
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.
```

Je li overhead i trošak ove tehnike vrijedan stvaranja čarobnijeg iskustva kupnje? Za mnoge online trgovce, apsolutno. Sudeći po svemu, trošak AI računanja će samo padati, posebno za pružatelje usluga hostinga open source modela u utrci prema dnu.

Koristite Predložak Uputa i StructuredIO zajedno s Ograđivanjem Odgovora za optimizaciju ovakve vrste chat završetka.

Prilagodljivi Redoslijed Polja

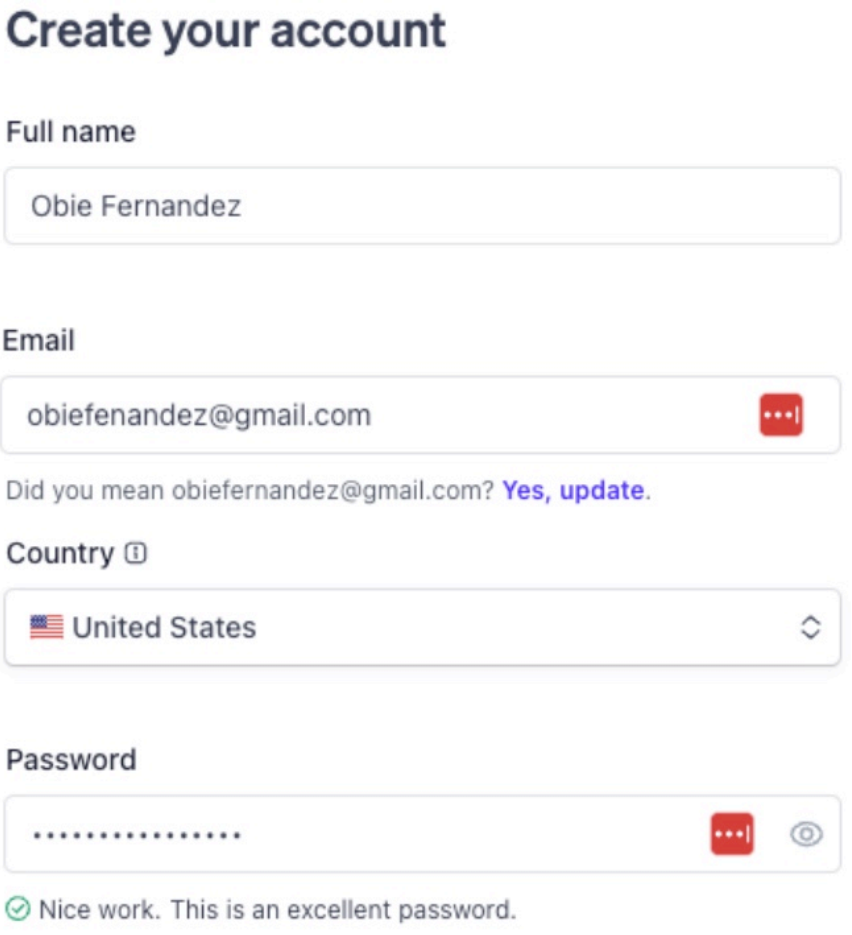
Redoslijed kojim se prikazuju polja obrasca može značajno utjecati na korisničko iskustvo i stopu završetka. S GenUI-jem, možete dinamički prilagoditi redoslijed polja na temelju korisničkog konteksta i važnosti svakog polja. Na primjer, ako korisnik ispunjava registracijski obrazac za fitness aplikaciju, obrazac može dati prioritet poljima vezanim uz njihove fitness ciljeve i preferencije, čineći proces relevantnijim i privlačnijim.

Personalizirani Mikrotekst

Upute, poruke o pogreškama i ostali mikrotekst povezan s obrascima također se može personalizirati pomoću GenUI-ja. Umjesto prikazivanja generičkih poruka o pogreškama poput "Nevažeća email adresa," možete generirati korisnije i kontekstualne poruke kao što je "Molimo unesite valjanu email adresu kako biste primili potvrdu narudžbe." Ovi personalizirani detalji mogu učiniti iskustvo ispunjavanja obrazaca pristupačnijim i manje frustrirajućim.

Personalizirana Validacija

U skladu s Personaliziranim Mikrotekstom, mogli biste koristiti AI za validaciju obrasca na način koji se čini čarobnim. Zamislite da pustite AI da validira obrazac korisničkog profila, tražeći potencijalne pogreške na *semantičkoj* razini.



Slika 9. Možete li uočiti semantičku validaciju koja se događa?

Progresivno Otkrivanje

GenUI može inteligentno odrediti koja su polja obrasca ključna na temelju korisničkog konteksta i postupno otkrivati dodatna polja prema potrebi. Ova tehnika progresivnog otkrivanja pomaže smanjiti kognitivno opterećenje i čini proces ispunjavanja obrazaca upravljivijim. Na primjer, ako se korisnik prijavljuje za osnovnu pretplatu, obrazac može

inicijalno prikazati samo ključna polja, a kako korisnik napreduje ili odabire određene opcije, dodatna relevantna polja mogu se dinamički uvoditi.

Kontekstualno Svjestan Tekst s Objašnjenjima

Oblačići s objašnjenjima često se koriste za pružanje dodatnih informacija ili smjernica korisnicima kada prelaze mišem preko ili komuniciraju s određenim elementima. S pristupom "Generiranje Kontekstualnog Sadržaja", možete generirati oblačiće s objašnjenjima koji se prilagođavaju korisničkom kontekstu i pružaju relevantne informacije. Na primjer, ako korisnik istražuje složenu funkcionalnost, oblačić može ponuditi personalizirane savjete ili primjere temeljene na njihovim prethodnim interakcijama ili razini vještine.

Tekst s objašnjenjima, poput uputa, opisa ili poruka za pomoć, može se dinamički generirati na temelju korisničkog konteksta. Umjesto predstavljanja generičkih objašnjenja, možete koristiti LLM-ove za generiranje teksta koji je prilagođen specifičnim potrebama ili pitanjima korisnika. Na primjer, ako korisnik ima poteškoća s određenim korakom u procesu, tekst s objašnjenjima može pružiti personalizirane smjernice ili savjete za rješavanje problema.

Mikrotekst se odnosi na male dijelove teksta koji vode korisnike kroz vašu aplikaciju, poput oznaka gumba, poruka o pogreškama ili upita za potvrdu. Primjenom pristupa Generiranje Kontekstualnog Sadržaja na mikrotekst, možete stvoriti prilagodljivo korisničko sučelje koje reagira na korisničke akcije i pruža relevantan i koristan tekst. Na primjer, ako se korisnik sprema izvršiti kritičnu radnju, upit za potvrdu može se dinamički generirati kako bi pružio jasnu i personaliziranu poruku.

Personalizirani tekst s objašnjenjima i oblačići mogu značajno poboljšati proces uvođenja novih korisnika. Pružanjem kontekstualno specifičnih smjernica i primjera, možete pomoći korisnicima da brzo razumiju i navigiraju aplikacijom, smanjujući krivulju učenja i povećavajući usvajanje.

Dinamički i kontekstualno svjesni elementi sučelja također mogu učiniti aplikaciju intuitivnijom i privlačnijom. Korisnici će vjerojatnije komunicirati s funkcionalnostima i istraživati ih kada je prateći tekst prilagođen njihovim specifičnim potrebama i interesima.

Do sada smo obradili ideje za poboljšanje postojećih UI paradigmi pomoću umjetne inteligencije, ali što je s ponovnim promišljanjem o tome kako se korisnička sučelja dizajniraju i implementiraju na radikalniji način?

Definiranje generativnog UI-ja

Za razliku od tradicionalnog UI dizajna, gdje dizajneri stvaraju fiksna, statična sučelja, GenUI ukazuje na budućnost u kojoj naš softver ima fleksibilna, personalizirana iskustva koja se mogu razvijati i prilagođavati u stvarnom vremenu. Svaki put kada koristimo konverzacijsko sučelje temeljeno na umjetnoj inteligenciji, omogućujemo UI-ju da se prilagodi posebnim potrebama korisnika. GenUI ide korak dalje primjenjujući tu razinu prilagodljivosti na *vizualno* sučelje softvera.

Razlog zašto je danas moguće eksperimentirati s GenUI idejama je taj što veliki jezični modeli već razumiju programiranje, a njihovo osnovno znanje uključuje UI tehnologije i okvire. Pitanje je, dakle, mogu li se veliki jezični modeli koristiti za generiranje UI elemenata, poput teksta, slika, rasporeda, pa čak i cijelih sučelja, koji su prilagođeni svakom pojedinom korisniku. Model bi mogao biti upućen da uzme u obzir različite faktore, poput korisnikovih prethodnih interakcija, izraženih preferencija, demografskih informacija i trenutnog konteksta uporabe, kako bi stvorio visoko personalizirana i relevantna sučelja.

GenUI se razlikuje od tradicionalnog dizajna korisničkog sučelja u nekoliko ključnih aspekata:

1. **Dinamično i prilagodljivo**: Tradicionalni UI dizajn uključuje stvaranje fiksnih, statičnih sučelja koja ostaju ista za sve korisnike. Nasuprot tome, GenUI omogućuje sučelja koja se mogu dinamički prilagođavati i mijenjati na temelju korisničkih potreba i konteksta. To znači da ista aplikacija može predstaviti različita sučelja različitim korisnicima ili čak istom korisniku u različitim situacijama.
2. **Personalizacija na velikoj skali**: Kod tradicionalnog dizajna, stvaranje personaliziranih iskustava za svakog korisnika često je nepraktično zbog potrebnog vremena i resursa. S druge strane, GenUI omogućuje personalizaciju na velikoj skali. Korištenjem UI-ja, dizajneri mogu stvoriti sučelja koja se automatski prilagođavaju jedinstvenim potrebama i preferencijama svakog korisnika, bez potrebe za ručnim dizajniranjem i razvijanjem zasebnih sučelja za svaki korisnički segment.
3. **Fokus na rezultate**: Tradicionalni UI dizajn često se fokusira na stvaranje vizualno privlačnih i funkcionalnih sučelja. Iako su ovi aspekti i dalje važni u GenUI-ju, primarni fokus se prebacuje prema postizanju željenih korisničkih rezultata. GenUI ima za cilj stvoriti sučelja koja su optimizirana za specifične ciljeve i zadatke svakog korisnika, dajući prioritet upotrebljivosti i učinkovitosti nad čisto estetskim razmatranjima.
4. **Kontinuirano učenje i poboljšanje**: GenUI sustavi mogu kontinuirano učiti i poboljšavati se tijekom vremena na temelju korisničkih interakcija i povratnih informacija. Dok korisnici koriste generirana sučelja, AI modeli mogu prikupljati podatke o korisničkom ponašanju, preferencijama i rezultatima, koristeći te informacije za usavršavanje i optimizaciju budućih generacija sučelja. Ovaj iterativni proces učenja omogućuje GenUI sustavima da tijekom vremena postaju sve učinkovitiji u zadovoljavanju korisničkih potreba.

Važno je napomenuti da GenUI nije isto što i alati za dizajn potpomognuti umjetnom inteligencijom, poput onih koji pružaju prijedloge ili automatiziraju određene zadatke dizajna. Iako ovi alati mogu biti korisni u pojednostavljivanju procesa dizajna, oni se i

dalje oslanjaju na dizajnere da donose konačne odluke i stvaraju statična sučelja. GenUI, s druge strane, uključuje aktivniju ulogu AI sustava u stvarnom generiranju i prilagodbi sučelja na temelju korisničkih podataka i konteksta.

GenUI predstavlja značajan pomak u načinu na koji pristupamo dizajnu korisničkog sučelja, odmičući se od rješenja koja odgovaraju svima prema visoko personaliziranim, prilagodljivim iskustvima. Korištenjem snage umjetne inteligencije, GenUI ima potencijal revolucionirati način na koji komuniciramo s digitalnim proizvodima i uslugama, stvarajući sučelja koja su intuitivnija, angažiranija i učinkovitija za svakog pojedinog korisnika.

Primjer

Za ilustraciju koncepta GenUI-ja, razmotrimo hipotetsku fitness aplikaciju pod nazivom "FitAI". Ova aplikacija ima za cilj pružiti personalizirane planove vježbanja i savjete o prehrani korisnicima na temelju njihovih individualnih ciljeva, razina kondicije i preferencija.

U tradicionalnom pristupu UI dizajnu, FitAI bi mogao imati fiksni set zaslona i elemenata koji su isti za sve korisnike. Međutim, s GenUI-jem, sučelje aplikacije moglo bi se dinamički prilagođavati jedinstvenim potrebama i kontekstu svakog korisnika.

Ovaj pristup je pomalo teško zamisliti za implementaciju u 2024. godini i možda čak nema adekvatan povrat ulaganja, ali je moguć.

Evo kako bi to moglo funkcionirati:

1. **Uvođenje**:
    - Umjesto standardnog upitnika, FitAI koristi konverzacijsku umjetnu inteligenciju za prikupljanje informacija o korisnikovim ciljevima, trenutnoj razini kondicije i preferencijama.

- Na temelju ove početne interakcije, UI generira personalizirani raspored nadzorne ploče, naglašavajući značajke i informacije najrelevantnije za korisnikove ciljeve.
- Trenutna AI tehnologija mogla bi imati na raspolaganju izbor komponenti zaslona za sastavljanje personalizirane nadzorne ploče.
- Buduća AI tehnologija mogla bi preuzeti ulogu iskusnog UI dizajnera i zapravo stvoriti nadzornu ploču *ispočetka*.

2. **Planer treninga**:

- AI prilagođava sučelje planera treninga specifično prema korisnikovoj razini iskustva i dostupnoj opremi.
- Za početnika bez opreme, mogao bi prikazivati jednostavne vježbe s vlastitom težinom uz detaljne upute i videe.
- Za naprednog korisnika s pristupom teretani, mogao bi prikazivati složenije rutine s manje pojašnjenja.
- Sadržaj planera treninga ne filtrira se jednostavno iz velikog nadskupa. Može se generirati *u trenutku* na temelju baze znanja koja se pretražuje s kontekstom koji uključuje sve što se zna o korisniku.

3. **Praćenje napretka**:

- Sučelje za praćenje napretka razvija se na temelju korisnikovih ciljeva i obrazaca angažmana.
- Ako je korisnik prvenstveno usredotočen na gubitak težine, sučelje bi moglo istaknuto prikazivati graf trenda težine i statistiku potrošnje kalorija.
- Za korisnika koji gradi mišićnu masu, moglo bi naglašavati napredak u snazi i promjene u sastavu tijela.
- AI može prilagoditi ovaj dio aplikacije stvarnom napretku korisnika. Ako napredak stane na određeno vrijeme, aplikacija može prijeći u način rada gdje pokušava navesti korisnika da otkrije razloge zastoja, kako bi ih ublažila.

4. **Savjeti o prehrani**:
    - Sekcija o prehrani prilagođava se korisnikovim prehrambenim preferencijama i ograničenjima.
    - Za veganskog korisnika, mogla bi prikazivati prijedloge biljnih obroka i izvore proteina.
    - Za korisnika s intolerancijom na gluten, automatski bi filtrirala namirnice koje sadrže gluten iz preporuka.
    - Ponovno, sadržaj se ne izvlači iz masivnog nadskupa podataka o obrocima koji vrijedi za sve korisnike, već se sintetizira iz baze znanja koja sadrži informacije prilagodljive specifičnoj situaciji i ograničenjima korisnika.
    - Na primjer, recepti se generiraju sa specifikacijama sastojaka koje odgovaraju konstantno promjenjivim kalorijskim potrebama korisnika kako se njihova razina kondicije i tjelesne statistike razvijaju.
5. **Motivacijski elementi**:
    - Motivacijski sadržaj i obavijesti aplikacije personalizirani su na temelju korisnikova tipa ličnosti i reakcije na različite motivacijske strategije.
    - Neki korisnici mogu primati ohrabrujuće poruke, dok drugi dobivaju povratne informacije više temeljene na podacima.

U ovom primjeru, GenUI omogućuje FitAI-u stvaranje visoko prilagođenog iskustva za svakog korisnika, potencijalno povećavajući angažman, zadovoljstvo i vjerojatnost postizanja fitness ciljeva. Elementi sučelja, sadržaj, pa čak i "osobnost" aplikacije prilagođavaju se kako bi najbolje služili potrebama i preferencijama svakog pojedinog korisnika.

Pomak prema dizajnu orijentiranom na ishode

GenUI predstavlja temeljni pomak u pristupu dizajnu korisničkog sučelja, prelazeći s fokusa na stvaranje specifičnih elemenata sučelja na cjelovitiji pristup orijentiran na ishode. Ovaj pomak ima nekoliko važnih implikacija:

1. **Fokus na korisničke ciljeve**:

    - Dizajneri će morati dublje razmišljati o korisničkim ciljevima i željenim ishodima umjesto o specifičnim komponentama sučelja.
    - Naglasak će biti na stvaranju sustava koji mogu generirati sučelja koja pomažu korisnicima da učinkovito postignu svoje ciljeve.
    - Pojavit će se novi UI okviri koji će AI dizajnerima dati alate potrebne za generiranje korisničkih iskustava *u trenutku* i *iz temelja* umjesto na temelju unaprijed definiranih specifikacija zaslona.

2. **Promjena uloge dizajnera**:

    - Dizajneri će prijeći s kreiranja fiksnih izgleda na definiranje pravila, ograničenja i smjernica koje AI sustavi trebaju slijediti pri generiranju sučelja.
    - Morat će razviti vještine u područjima poput analize podataka, inženjerstva uputa za AI i sistemskog razmišljanja kako bi učinkovito vodili GenUI sustave.

3. **Važnost korisničkog istraživanja**:

    - Korisničko istraživanje postaje još kritičnije u GenUI kontekstu, jer dizajneri trebaju razumjeti ne samo korisničke preferencije, već i kako se te preferencije i potrebe mijenjaju u različitim kontekstima.
    - Kontinuirano korisničko testiranje i povratne petlje bit će ključni za usavršavanje i poboljšanje sposobnosti AI-ja da generira učinkovita sučelja.

4. **Dizajniranje za varijabilnost**:

    - Umjesto stvaranja jednog "savršenog" sučelja, dizajneri će morati razmotriti više mogućih varijacija i osigurati da sustav može generirati odgovarajuća sučelja za različite korisničke potrebe.

- To uključuje dizajniranje za granične slučajeve i osiguravanje da generirana sučelja održavaju upotrebljivost i pristupačnost kroz različite konfiguracije.
- Diferencijacija proizvoda poprima nove dimenzije koje uključuju različite perspektive o psihologiji korisnika i iskorištavanje jedinstvenih skupova podataka i baza znanja nedostupnih konkurentima.

Izazovi i razmatranja

Iako GenUI nudi uzbudljive mogućnosti, također predstavlja nekoliko izazova i razmatranja:

1. **Tehnička ograničenja**:

    - Trenutna AI tehnologija, iako napredna, još uvijek ima ograničenja u razumijevanju složenih korisničkih namjera i generiranju istinski kontekstualno svjesnih sučelja.
    - Problemi s performansama vezani uz generiranje elemenata sučelja u stvarnom vremenu, posebno na manje snažnim uređajima.

2. **Zahtjevi za podacima**:

    - Ovisno o slučaju uporabe, učinkoviti GenUI sustavi mogu zahtijevati značajne količine korisničkih podataka za generiranje personaliziranih sučelja.
    - Izazovi u etičkom prikupljanju autentičnih korisničkih podataka izazivaju zabrinutost oko privatnosti i sigurnosti podataka, kao i potencijalnih pristranosti u podacima koji se koriste za treniranje GenUI modela.

3. **Upotrebljivost i konzistentnost**:

- Barem dok praksa ne postane raširena, aplikacija s konstantno promjenjivim sučeljima mogla bi dovesti do problema s upotrebljivošću, jer bi korisnici mogli imati poteškoća s pronalaženjem poznatih elemenata ili učinkovitom navigacijom.
- Ključno će biti postizanje ravnoteže između personalizacije i održavanja konzistentnog sučelja koje se može naučiti.

4. **Pretjerano oslanjanje na UI**:

    - Postoji rizik od pretjeranog delegiranja dizajnerskih odluka AI sustavima, što potencijalno može dovesti do neinspiriranih, problematičnih ili jednostavno nefunkcionalnih izbora sučelja.
    - Ljudski nadzor i mogućnost nadjačavanja AI-generiranih dizajna ostat će važni u doglednoj budućnosti.

5. **Pitanja pristupačnosti**:

    - Osiguravanje da dinamički generirana sučelja ostanu pristupačna korisnicima s invaliditetom predstavlja potpuno nove izazove, što je zabrinjavajuće s obzirom na lošu razinu usklađenosti s pristupačnošću koju pokazuju tipični sustavi.
    - S druge strane, AI dizajneri mogu biti implementirani s *ugrađenom* brigom za pristupačnost i mogućnostima za izgradnju pristupačnih sučelja u hodu, baš kao što grade UI za korisnike bez poteškoća.
    - U svakom slučaju, GenUI sustavi trebali bi biti dizajnirani s robusnim smjernicama za pristupačnost i procesima testiranja.

6. **Povjerenje korisnika i transparentnost**:

    - Korisnici se mogu osjećati nelagodno sa sučeljima koja "znaju previše" o njima ili se mijenjaju na načine koje ne razumiju.
    - Pružanje transparentnosti o tome kako i zašto se sučelja personaliziraju bit će važno za izgradnju povjerenja korisnika.

Budući izgledi i mogućnosti

Budućnost Generativnog UI-ja (GenUI) obećava revoluciju u načinu na koji komuniciramo s digitalnim proizvodima i uslugama. Kako se ova tehnologija nastavlja razvijati, možemo očekivati tektonski pomak u načinu na koji se korisnička sučelja dizajniraju, implementiraju i doživljavaju. Smatram da je GenUI fenomen koji će konačno gurnuti naš softver u područje onoga što se danas smatra znanstvenom fantastikom.

Jedna od najuzbudljivijih perspektiva GenUI-ja je njegov potencijal za unapređenje pristupačnosti u razmjerima koji nadilaze samo osiguravanje da ljudi s ozbiljnim invaliditetom nisu potpuno isključeni iz korištenja vašeg softvera. Automatskim prilagođavanjem sučelja individualnim potrebama korisnika, GenUI bi mogao učiniti digitalna iskustva inkluzivnijima nego ikad prije. Zamislite sučelja koja se besprijekorno prilagođavaju pružajući veći tekst za mlađe ili korisnike s oštećenim vidom ili pojednostavljene rasporede za one s kognitivnim poteškoćama, sve bez potrebe za ručnom konfiguracijom ili zasebnim "pristupačnim" verzijama aplikacija.

Mogućnosti personalizacije GenUI-ja vjerojatno će potaknuti povećanu angažiranost korisnika, zadovoljstvo i lojalnost kroz širok raspon digitalnih proizvoda. Kako sučelja postaju više usklađena s individualnim preferencijama i ponašanjima, korisnici će pronaći digitalna iskustva intuitivnijima i ugodnijima, što potencijalno vodi do dubljih i značajnijih interakcija s tehnologijom.

GenUI također ima potencijal transformirati proces uvođenja novih korisnika. Stvaranjem intuitivnih, personaliziranih iskustava za nove korisnike koja se brzo prilagođavaju razini stručnosti svakog korisnika, GenUI bi mogao značajno smanjiti krivulju učenja povezanu s novim aplikacijama. To bi moglo dovesti do bržih stopa usvajanja i povećanog samopouzdanja korisnika u istraživanju novih značajki i funkcionalnosti.

Još jedna uzbudljiva mogućnost je sposobnost GenUI-ja da održi konzistentno

korisničko iskustvo na različitim uređajima i platformama, optimizirajući za svaki specifični kontekst uporabe. To bi moglo riješiti dugogodišnji izazov pružanja koherentnih iskustava kroz sve fragmentiraniji krajolik uređaja, od pametnih telefona i tableta do stolnih računala i tehnologija u nastajanju poput naočala za proširenu stvarnost.

Priroda GenUI-ja temeljena na podacima otvara mogućnosti za brzu iteraciju i poboljšanje u dizajnu korisničkog sučelja. Prikupljanjem podataka u stvarnom vremenu o tome kako korisnici komuniciraju s generiranim sučeljima, dizajneri i programeri mogu dobiti neviđene uvide u ponašanje i preferencije korisnika. Ova povratna petlja mogla bi dovesti do kontinuiranih poboljšanja u dizajnu UI-ja, vođenih stvarnim obrascima korištenja umjesto pretpostavkama ili ograničenim korisničkim testiranjem.

Za pripremu za ovu promjenu, dizajneri će morati razviti svoje vještine i način razmišljanja. Fokus će se pomaknuti s kreiranja fiksnih rasporeda na razvoj sveobuhvatnih dizajn sustava i smjernica koje mogu informirati generiranje sučelja vođeno AI-jem. Dizajneri će morati razviti duboko razumijevanje analize podataka, AI tehnologija i sustavnog razmišljanja kako bi učinkovito vodili GenUI sustave.

Štoviše, kako GenUI zamagljuje granice između dizajna i tehnologije, dizajneri će morati uže surađivati s programerima i znanstvenicima podataka. Ovaj interdisciplinarni pristup bit će ključan u stvaranju GenUI sustava koji nisu samo vizualno privlačni i user-friendly, već i tehnički robusni i etički ispravni.

Etičke implikacije GenUI-ja također će doći u prvi plan kako tehnologija sazrijeva. Dizajneri će igrati ključnu ulogu u razvoju okvira za odgovornu upotrebu umjetne inteligencije u dizajnu sučelja, osiguravajući da personalizacija poboljšava korisničko iskustvo bez ugrožavanja privatnosti ili neetičke manipulacije ponašanjem korisnika.

Gledajući prema budućnosti, GenUI predstavlja i uzbudljive prilike i značajne izazove. Ima potencijal stvoriti intuitivnija, učinkovitija i zadovoljavajuća digitalna iskustva za korisnike diljem svijeta. Iako će zahtijevati od dizajnera da se prilagode i steknu nove vještine, također pruža neviđenu priliku za oblikovanje budućnosti interakcije čovjeka

i računala na dubok i smislen način. Put prema potpuno realiziranim GenUI sustavima nedvojbeno će biti složen, ali potencijalne nagrade u smislu poboljšanog korisničkog iskustva i digitalne pristupačnosti čine ga budućnošću za koju se vrijedi zalagati.

Inteligentno orkestriranje tijeka rada

U području razvoja aplikacija, tijekovi rada igraju ključnu ulogu u definiranju načina strukturiranja i izvršavanja zadataka, procesa i korisničkih interakcija. Kako aplikacije postaju sve složenije, a očekivanja korisnika nastavljaju rasti, potreba za inteligentnim i prilagodljivim orkestriranjem tijeka rada postaje sve očitija.

Pristup "Inteligentnog orkestriranja tijeka rada" fokusira se na korištenje AI komponenti za dinamičko orkestriranje i optimizaciju složenih tijekova rada unutar aplikacija. Cilj je stvoriti aplikacije koje su učinkovitije, responzivnije i prilagodljivije stvarnovremenskim podacima i kontekstu.

U ovom poglavlju istražit ćemo ključne principe i obrasce koji podupiru pristup inteligentnog orkestriranja tijeka rada. Razmotrit ćemo kako se AI može koristiti za

inteligentno usmjeravanje zadataka, automatizaciju odlučivanja i dinamičku prilagodbu tijekova rada na temelju različitih faktora kao što su ponašanje korisnika, performanse sustava i poslovna pravila. Kroz praktične primjere i scenarije iz stvarnog svijeta, demonstrirat ćemo transformativni potencijal AI-ja u pojednostavljivanju i optimizaciji tijekova rada aplikacija.

Bilo da gradite poslovne aplikacije sa složenim poslovnim procesima ili aplikacije namijenjene krajnjim korisnicima s dinamičkim korisničkim putovanjima, obrasci i tehnike o kojima se raspravlja u ovom poglavlju opremit će vas znanjem i alatima za stvaranje inteligentnih i učinkovitih tijekova rada koji poboljšavaju cjelokupno korisničko iskustvo i stvaraju poslovnu vrijednost.

Poslovna potreba

Tradicionalni pristupi upravljanju tijekom rada često se oslanjaju na unaprijed definirane pravila i statička stabla odlučivanja, koja mogu biti kruta, nefleksibilna i nesposobna nositi se s dinamičkom prirodom modernih aplikacija.

Razmotrite scenarij gdje aplikacija za e-trgovinu treba upravljati složenim procesom ispunjenja narudžbe. Tijek rada može uključivati više koraka kao što su validacija narudžbe, provjera zaliha, obrada plaćanja, dostava i obavijesti kupcima. Svaki korak može imati vlastiti skup pravila, ovisnosti, vanjske integracije i mehanizme za rukovanje iznimkama. Upravljanje takvim tijekom rada ručno ili kroz tvrdo kodirane logike može brzo postati nezgrapno, sklono pogreškama i teško za održavanje.

Štoviše, kako aplikacija skalira i broj istovremenih korisnika raste, tijek rada možda će se trebati prilagoditi i optimizirati na temelju podataka u stvarnom vremenu i performansi sustava. Na primjer, tijekom razdoblja vršnog prometa, aplikacija će možda trebati dinamički prilagoditi tijek rada kako bi prioritizirala određene zadatke, učinkovito rasporedila resurse i osigurala nesmetano korisničko iskustvo.

Tu nastupa pristup “Inteligentnog orkestriranja tijeka rada”. Korištenjem AI

komponenti, razvojni programeri mogu stvoriti tijekove rada koji su inteligentni, prilagodljivi i samoptimizirajući. AI može analizirati ogromne količine podataka, učiti iz prošlih iskustava i donositi informirane odluke u stvarnom vremenu za učinkovito orkestriranje tijeka rada.

Ključne prednosti

1. **Povećana učinkovitost**: AI može optimizirati raspodjelu zadataka, korištenje resursa i izvršavanje tijeka rada, što dovodi do bržih vremena obrade i poboljšane ukupne učinkovitosti.
2. **Prilagodljivost**: Tijekovi rada vođeni AI-jem mogu se dinamički prilagoditi promjenjivim uvjetima, kao što su fluktuacije u korisničkoj potražnji, performansama sustava ili poslovnim zahtjevima, osiguravajući da aplikacija ostane responzivna i otporna.
3. **Automatizirano odlučivanje**: AI može automatizirati složene procese odlučivanja unutar tijeka rada, smanjujući ručne intervencije i minimizirajući rizik ljudskih pogrešaka.
4. **Personalizacija**: AI može analizirati ponašanje korisnika, preferencije i kontekst kako bi personalizirao tijek rada i isporučio prilagođena iskustva pojedinačnim korisnicima.
5. **Skalabilnost**: Tijekovi rada pokretani AI-jem mogu se besprijekorno skalirati kako bi rukovali sve većim količinama podataka i korisničkih interakcija, bez ugrožavanja performansi ili pouzdanosti.

U sljedećim odjeljcima istražit ćemo ključne obrasce i tehnike koje omogućuju implementaciju inteligentnih tijekova rada i prikazati primjere iz stvarnog svijeta kako AI transformira upravljanje tijekovima rada u modernim aplikacijama.

Ključni obrasci

Za implementaciju inteligentnog orkestriranja tijeka rada u aplikacijama, razvojni programeri mogu iskoristiti nekoliko ključnih obrazaca koji koriste snagu AI-ja. Ovi obrasci pružaju strukturirani pristup dizajniranju i upravljanju tijekovima rada, omogućujući aplikacijama da se prilagode, optimiziraju i automatiziraju procese na temelju podataka i konteksta u stvarnom vremenu. Istražimo neke od temeljnih obrazaca u inteligentnom orkestriranju tijeka rada.

Dinamičko usmjeravanje zadataka

Ovaj obrazac uključuje korištenje AI-ja za inteligentno usmjeravanje zadataka unutar tijeka rada na temelju različitih faktora kao što su prioritet zadatka, dostupnost resursa i performanse sustava. AI algoritmi mogu analizirati karakteristike svakog zadatka, razmotriti trenutno stanje sustava i donositi informirane odluke za dodjeljivanje zadataka najprikladnijim resursima ili putovima obrade. Dinamičko usmjeravanje zadataka osigurava da su zadaci učinkovito distribuirani i izvršeni, optimizirajući ukupne performanse tijeka rada.

```
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
```

Primjetite petlju stvorenu `while` izrazom u retku 29, koja nastavlja slati upite AI-u sve dok zadatak nije dodijeljen. U retku 35 spremamo transkript zadatka za kasniju analizu i debugiranje, ako to postane potrebno.

Kontekstualno odlučivanje

Možete koristiti vrlo sličan kod za donošenje odluka temeljenih na kontekstu unutar tijeka rada. Analiziranjem relevantnih podataka poput korisničkih preferencija, povijesnih obrazaca i podataka u stvarnom vremenu, AI komponente mogu odrediti najprikladniji smjer djelovanja u svakoj točki odlučivanja u tijeku rada. Prilagodite ponašanje vašeg tijeka rada na temelju specifičnog konteksta svakog korisnika ili scenarija, pružajući personalizirano i optimizirano iskustvo.

Adaptivna kompozicija tijeka rada

Ovaj obrazac se fokusira na dinamičko sastavljanje i prilagođavanje tijekova rada na temelju promjenjivih zahtjeva ili uvjeta. AI može analizirati trenutno stanje tijeka rada, identificirati uska grla ili neučinkovitosti te automatski modificirati strukturu tijeka rada kako bi optimizirao performanse. Adaptivna kompozicija tijeka rada omogućuje aplikacijama da se kontinuirano razvijaju i poboljšavaju svoje procese bez potrebe za ručnom intervencijom.

Rukovanje iznimkama i oporavak

Rukovanje iznimkama i oporavak su ključni aspekti inteligentne orkestracije tijeka rada. Kada radite s AI komponentama i složenim tijekovima rada, ključno je predvidjeti i elegantno rukovati iznimkama kako bi se osigurala stabilnost i pouzdanost sustava.

Evo nekoliko ključnih razmatranja i tehnika za rukovanje iznimkama i oporavak u inteligentnim tijekovima rada:

1. **Propagacija iznimki**: Implementirajte dosljedan pristup za propagaciju iznimki kroz komponente tijeka rada. Kada se iznimka pojavi unutar komponente, treba je uhvatiti, zabilježiti i propagirati do orkestratora ili zasebne komponente odgovorne za rukovanje iznimkama. Ideja je centralizirati rukovanje iznimkama i spriječiti tiho gutanje iznimki, kao i otvoriti mogućnosti za Inteligentno rukovanje pogreškama.
2. **Mehanizmi ponovnih pokušaja**: Mehanizmi ponovnih pokušaja pomažu poboljšati otpornost tijeka rada i elegantno rukovati povremenim neuspjesima. Svakako pokušajte implementirati mehanizme ponovnih pokušaja za prolazne ili oporavive iznimke, poput mrežne povezivosti ili nedostupnosti resursa koji se mogu automatski ponovno pokušati nakon određenog kašnjenja. Imati AI-pogonjen orkestrator ili rukovatelj iznimkama znači da vaše strategije ponovnih

pokušaja ne moraju biti mehaničke prirode, oslanjajući se na fiksne algoritme poput eksponencijalnog povlačenja. Možete prepustiti rukovanje ponovnim pokušajima "diskreciji" AI komponente odgovorne za odlučivanje kako rukovati iznimkom.

3. **Strategije oporavka**: Ako AI komponenta ne uspije pružiti valjani odgovor ili naiđe na pogrešku—što je česta pojava s obzirom na njenu cutting-edge prirodu—imajte mehanizam oporavka koji će osigurati da se tijek rada može nastaviti. To može uključivati korištenje zadanih vrijednosti, alternativnih algoritama ili Čovjeka u petlji za donošenje odluka i održavanje tijeka rada.
4. **Kompenzirajuće akcije**: Direktive orkestratora trebaju uključivati upute o kompenzirajućim akcijama za rukovanje iznimkama koje se ne mogu automatski riješiti. Kompenzirajuće akcije su koraci poduzeti za poništavanje ili ublažavanje učinaka neuspjele operacije. Na primjer, ako korak obrade plaćanja ne uspije, kompenzirajuća akcija mogla bi biti povrat transakcije i obavještavanje korisnika. Kompenzirajuće akcije pomažu održati konzistentnost i integritet podataka u slučaju iznimki.
5. **Nadzor iznimki i upozoravanje**: Postavite mehanizme nadzora i upozoravanja za otkrivanje i obavještavanje relevantnih dionika o kritičnim iznimkama. Orkestrator može biti svjestan pragova i pravila za pokretanje upozorenja kada iznimke premaše određene granice ili kada se pojave određene vrste iznimki. To omogućuje proaktivnu identifikaciju i rješavanje problema prije nego što utječu na cjelokupni sustav.

Evo primjera rukovanja iznimkama i oporavka u Ruby komponenti tijeka rada:

```ruby
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 primjeru, komponenta `InventoryManager` provjerava dostupnost proizvoda za određenu narudžbu. Ako je dostupna količina nedovoljna, podiže se `InsufficientInventoryError`. Iznimka se hvata, bilježi, i implementira se mehanizam ponovnog pokušaja. Ako se prekorači ograničenje broja pokušaja, komponenta prelazi na ručnu intervenciju obavještavanjem administratora.

Implementacijom robusnih mehanizama za rukovanje i oporavak od iznimaka, možete osigurati da su vaši inteligentni tijekovi rada otporni, održivi i sposobni elegantno se

nositi s neočekivanim situacijama.

Ovi obrasci čine temelj orkestracije inteligentnog tijeka rada i mogu se kombinirati i prilagoditi specifičnim zahtjevima različitih aplikacija. Korištenjem ovih obrazaca, programeri mogu stvoriti tijekove rada koji su fleksibilni, otporni i optimizirani za performanse i korisničko iskustvo.

U sljedećem odjeljku, istražit ćemo kako se ovi obrasci mogu implementirati u praksi, koristeći primjere iz stvarnog svijeta i isječke koda kako bismo ilustrirali integraciju AI komponenti u upravljanje tijekom rada.

Implementacija orkestracije inteligentnog tijeka rada u praksi

Sada kada smo istražili ključne obrasce u orkestraciji inteligentnog tijeka rada, zaronimo u to kako se ovi obrasci mogu implementirati u aplikacijama iz stvarnog svijeta. Pružit ćemo praktične primjere i isječke koda kako bismo ilustrirali integraciju AI komponenti u upravljanje tijekom rada.

Inteligentni procesor narudžbi

Zaronimo u praktični primjer implementacije orkestracije inteligentnog tijeka rada koristeći AI-pogonljenu komponentu `OrderProcessor` u Ruby on Rails e-commerce aplikaciji. `OrderProcessor` realizira koncept Upravitelja procesa za integraciju poduzeća koji smo prvi put susreli u Poglavlju 3 kada smo raspravljali o Mnoštvu radnika. Komponenta će biti odgovorna za upravljanje tijekom izvršavanja narudžbe, donošenje odluka o usmjeravanju na temelju međurezultata i orkestraciju izvršavanja različitih koraka obrade.

Proces izvršavanja narudžbe uključuje više koraka kao što su validacija narudžbe, provjera zaliha, obrada plaćanja i otprema. Svaki korak je implementiran kao zaseban radni proces koji izvršava specifični zadatak i vraća rezultat `OrderProcessor`-u. Koraci nisu obavezni i ne moraju se nužno izvršavati točno određenim redoslijedom.

Evo primjera implementacije `OrderProcessor`-a. Sadrži dva mixina iz Raixa. Prvi (`ChatCompletion`) daje mu mogućnost chat završavanja, što ga čini AI komponentom. Drugi (`FunctionDispatch`) omogućuje pozivanje funkcija od strane AI-a, dopuštajući mu da odgovori na upit pozivom funkcije umjesto tekstualnom porukom.

Radne funkcije (`validate_order`, `check_inventory`, i druge) delegiraju svojim odgovarajućim radnim klasama, koje mogu biti AI ili ne-AI komponente, s jedinim zahtjevom da vrate rezultate svog rada u formatu koji se može predstaviti kao string.

Kao i sa svim ostalim primjerima u ovom dijelu knjige, ovaj kod je praktički pseudo-kod i namijenjen je samo prenošenju značenja obrasca i inspiriranju vaših vlastitih kreacija. Potpuni opisi obrazaca i cjeloviti primjeri koda uključeni su u Dijelu 2.

```
1  class OrderProcessor
2    include Raix::ChatCompletion
3    include Raix::FunctionDispatch
4
5    SYSTEM_DIRECTIVE = "You are an order processor, tasked with..."
6
7    def initialize(order)
8      self.order = order
9      transcript << { system: SYSTEM_DIRECTIVE }
10     transcript << { user: order.to_json }
11   end
12
13   def perform
14     # will continue looping until `stop_looping!` is called
15     chat_completion(loop: true)
16   end
17
```

```
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 primjeru, OrderProcessor je inicijaliziran s objektom narudžbe i održava prijepis izvršavanja tijeka rada, u tipičnom formatu prijepisa razgovora koji je svojstven velikim jezičnim modelima. AI-ju je dana potpuna kontrola za orkestracijom izvršavanja različitih koraka obrade, kao što su validacija narudžbe, provjera zaliha, obrada plaćanja i dostava.

Svaki put kada se pozove metoda `chat_completion`, prijepis se šalje AI-ju kako bi pružio završetak u obliku poziva funkcije. U potpunosti je na AI-ju da analizira rezultat prethodnog koraka i odredi odgovarajuću akciju. Na primjer, ako provjera zaliha otkrije nisku razinu zaliha, `OrderProcessor` može zakazati zadatak nadopune. Ako obrada plaćanja ne uspije, može pokrenuti ponovni pokušaj ili obavijestiti korisničku podršku.

> Gore navedeni primjer nema definirane funkcije za nadopunu ili obavještavanje korisničke podrške, ali apsolutno bi mogao imati.

Prijepis raste svaki put kada se pozove funkcija i služi kao zapis izvršavanja tijeka rada, uključujući rezultate svakog koraka i upute generirane od strane AI-ja za sljedeće korake. Ovaj prijepis može se koristiti za otklanjanje pogrešaka, reviziju i pružanje uvida u proces ispunjavanja narudžbe.

Korištenjem AI-ja u `OrderProcessor`-u, e-commerce aplikacija može dinamički prilagoditi tijek rada na temelju podataka u stvarnom vremenu i inteligentno upravljati iznimkama. AI komponenta može donositi informirane odluke, optimizirati tijek rada i osigurati nesmetanu obradu narudžbi čak i u složenim scenarijima.

Činjenica da je jedini zahtjev za radne procese vraćanje razumljivog izlaza koji AI može razmotriti pri odlučivanju što dalje učiniti, možda vam počne ukazivati na to kako ovaj pristup može smanjiti posao mapiranja ulaza/izlaza koji je tipično uključen pri

integraciji različitih sustava međusobno.

Inteligentni moderator sadržaja

Aplikacije društvenih mreža općenito zahtijevaju barem minimalno moderiranje sadržaja kako bi se osigurala sigurna i zdrava zajednica. Ovaj primjer komponente `ContentModerator` koristi AI za inteligentno orkestriranje tijeka moderiranja, donoseći odluke na temelju karakteristika sadržaja i rezultata različitih koraka moderiranja.

Proces moderiranja uključuje više koraka kao što su analiza teksta, prepoznavanje slika, procjena reputacije korisnika i ručni pregled. Svaki korak je implementiran kao zaseban radni proces koji izvršava određeni zadatak i vraća rezultat `ContentModerator`-u.

Evo primjera 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

  def model
    "openai/gpt-4"
  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)
62      continue_with(result)
63    end
64
```

```
  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 primjeru, `ContentModerator` je inicijaliziran s objektom sadržaja i održava zapis moderiranja u formatu razgovora. AI komponenta ima potpunu kontrolu nad tijekom moderiranja, odlučujući koje korake izvršiti na temelju karakteristika sadržaja i rezultata svakog koraka.

Dostupne radne funkcije koje AI može pozvati 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 zapis moderiranja, s iznimkom funkcije eskalacije koja djeluje kao završno stanje. Konačno, funkcije `approve_content` i `reject_content` također djeluju 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 vizualnom pregledu. Ako bilo koji radni proces upozori na potencijalno štetan sadržaj, AI može odlučiti eskalirati sadržaj na ručni pregled ili ga odmah odbiti. No ovisno o ozbiljnosti upozorenja, AI može odlučiti koristiti rezultate procjene reputacije korisnika pri odlučivanju kako postupiti sa sadržajem u kojem nije siguran. Ovisno o slučaju uporabe, možda pouzdani korisnici imaju više slobode u onome što mogu objaviti. I tako dalje…

Kao i u prethodnom primjeru upravitelja procesa, zapis moderiranja služi kao evidencija izvršavanja tijeka rada, uključujući rezultate svakog koraka i odluke koje je generirao AI. Ovaj zapis može se koristiti za reviziju, transparentnost i poboljšanje procesa moderiranja tijekom vremena.

Korištenjem AI-ja u `ContentModerator`-u, aplikacija društvenih medija može dinamički prilagoditi tijek moderiranja na temelju karakteristika sadržaja i inteligentno upravljati složenim scenarijima moderiranja. AI komponenta može donositi informirane odluke, optimizirati tijek rada i osigurati sigurno i zdravo iskustvo zajednice.

Istražimo još dva primjera koji demonstriraju prediktivno raspoređivanje zadataka te rukovanje iznimkama i oporavak u kontekstu inteligentne orkestracije tijeka rada.

Prediktivno raspoređivanje zadataka u sustavu korisničke podrške

U aplikaciji za korisničku podršku izrađenoj pomoću Ruby on Rails, učinkovito upravljanje i određivanje prioriteta korisničkih prijava ključno je za pružanje pravovremene pomoći korisnicima. Komponenta `SupportTicketScheduler` koristi

AI za prediktivno raspoređivanje i dodjeljivanje korisničkih prijava dostupnim agentima na temelju različitih faktora kao što su hitnost prijave, 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

```

```
80    def continue_with(result)
81      @transcript << { function: result }
82      complete(@transcript)
83    end
84  end
```

U ovom primjeru, `SupportTicketScheduler` je inicijaliziran s objektom tiketa za podršku i održava zapisnik raspoređivanja. AI komponenta analizira detalje tiketa i prediktivno raspoređuje dodjelu tiketa na temelju faktora poput hitnosti tiketa, stručnosti agenta i predviđenog radnog opterećenja agenta.

Dostupne funkcije koje AI može pozvati 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 te dodaje rezultat u zapisnik raspoređivanja. AI također ima mogućnost odgoditi dodjelu pomoću funkcije `delay_assignment`.

AI komponenta pregledava zapisnik raspoređivanja i donosi informirane odluke o dodjeli tiketa. Uzima u obzir hitnost tiketa, stručnost dostupnih agenata i predviđeno radno opterećenje svakog agenta kako bi odredila najprikladnijeg agenta za rješavanje tiketa.

Korištenjem prediktivnog raspoređivanja zadataka, aplikacija za korisničku podršku može optimizirati dodjelu tiketa, smanjiti vrijeme odziva i poboljšati ukupno zadovoljstvo korisnika. Proaktivno i učinkovito upravljanje tiketima za podršku osigurava da se pravi tiketi dodjeljuju pravim agentima u pravo vrijeme.

Upravljanje iznimkama i oporavak u cjevovodu za obradu podataka

Upravljanje iznimkama i oporavak od grešaka ključni su za osiguravanje integriteta podataka i sprječavanje gubitka podataka. Komponenta `DataProcessingOrchestrator` koristi AI za inteligentno upravljanje iznimkama i orkestriranje procesa oporavka u cjevovodu za obradu 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 primjeru, `DataProcessingOrchestrator` je inicijaliziran s objektom grupe podataka i održava transkript obrade. AI komponenta upravlja tijekom obrade podataka, upravlja iznimkama i oporavlja se od grešaka prema potrebi.

Dostupne funkcije koje AI može pozvati uključuju `validate_data`, `process_data`, `request_fix`,`retry_processing` `imark_data_as_failed`. Svaka funkcija delegira zadatak odgovarajućoj komponenti za obradu podataka i dodaje rezultat ili detalje o iznimci u transkript obrade.

Ako se tijekom koraka `validate_data` pojavi iznimka pri validaciji, funkcija `handle_validation_exception` dodaje podatke o iznimci u transkript i vraća kontrolu AI-ju. Slično tome, ako se tijekom koraka `process_data` pojavi iznimka pri obradi, AI može odlučiti o strategiji oporavka.

Ovisno o prirodi nastale iznimke, AI može prema vlastitoj procjeni odlučiti pozvati `request_fix`, koji delegira AI-pokrenutoj komponenti `SmartDataFixer` (pogledajte poglavlje o Samoobnavljajućim podacima). Program za popravak podataka dobiva

opis na običnom hrvatskom jeziku o tome kako bi trebao modificirati `@data_batch` kako bi se obrada mogla ponovno pokušati. Možda bi uspješno ponovno pokretanje podrazumijevalo uklanjanje zapisa iz grupe podataka koji nisu prošli validaciju i/ili njihovo kopiranje u drugi tijek obrade za ljudski pregled? Mogućnosti su gotovo beskonačne.

Uključivanjem AI-vođenog upravljanja iznimkama i oporavka, aplikacija za obradu podataka postaje otpornija i tolerantnija na pogreške. `DataProcessingOrchestrator` inteligentno upravlja iznimkama, minimizira gubitak podataka i osigurava neometano izvršavanje tijeka obrade podataka.

Nadzor i bilježenje

Nadzor i bilježenje pružaju uvid u napredak, performanse i zdravlje komponenti tijeka rada pokrenutih AI-jem, omogućujući programerima praćenje i analizu ponašanja sustava. Implementacija učinkovitih mehanizama nadzora i bilježenja ključna je za otklanjanje grešaka, reviziju i kontinuirano poboljšanje inteligentnih tijekova rada.

Nadzor napretka i performansi tijeka rada

Kako bi se osiguralo neometano izvršavanje inteligentnih tijekova rada, važno je nadzirati napredak i performanse svake komponente tijeka rada. To uključuje praćenje ključnih metrika i događaja tijekom životnog ciklusa tijeka rada.

Neki važni aspekti za nadzor uključuju:

1. **Vrijeme izvršavanja tijeka rada**: Mjerenje vremena koje je potrebno svakoj komponenti tijeka rada da dovrši svoj zadatak. Ovo pomaže u identificiranju uskih grla performansi i optimizaciji ukupne učinkovitosti tijeka rada.

2. **Iskorištenost resursa**: Praćenje iskorištenosti sistemskih resursa, poput CPU-a, memorije i pohrane, od strane svake komponente tijeka rada. Ovo pomaže osigurati da sustav radi unutar svojih kapaciteta i može učinkovito upravljati radnim opterećenjem.

3. **Stope pogrešaka i iznimke**: Praćenje pojave pogrešaka i iznimki unutar komponenti tijeka rada. Ovo pomaže u identificiranju potencijalnih problema i omogućuje proaktivno upravljanje pogreškama i oporavak.

4. **Točke odlučivanja i ishodi**: Nadzor točaka odlučivanja unutar tijeka rada i ishoda odluka koje donosi AI. Ovo pruža uvid u ponašanje i učinkovitost AI komponenti.

Podaci prikupljeni procesima nadzora mogu se prikazati na kontrolnim pločama ili koristiti kao ulazni podaci za planirane izvještaje koji informiraju administratore sustava o zdravlju sustava.

Podaci nadzora mogu se proslijediti AI-pokrenutom procesu administratora sustava na pregled i potencijalnu akciju!

Bilježenje ključnih događaja i odluka

Bilježenje je ključna praksa koja uključuje hvatanje i pohranjivanje relevantnih informacija o ključnim događajima, odlukama i iznimkama koje se pojavljuju tijekom izvršavanja tijeka rada.

Neki važni aspekti za bilježenje uključuju:

1. **Pokretanje i završetak tijeka rada**: Bilježenje vremena početka i završetka svake instance tijeka rada, zajedno s relevantnim metapodacima poput ulaznih podataka i korisničkog konteksta.

2. **Izvršavanje komponenti**: Bilježenje detalja izvršavanja svake komponente tijeka rada, uključujući ulazne parametre, izlazne rezultate i sve generirane međupodatke.

3. **AI odluke i obrazloženja**: Bilježenje odluka koje donose AI komponente, zajedno s pripadajućim obrazloženjem ili ocjenama pouzdanosti. Ovo pruža transparentnost i omogućuje reviziju odluka koje donosi AI.

4. **Iznimke i poruke o pogreškama**: Bilježenje svih iznimki ili poruka o pogreškama na koje se naiđe tijekom izvršavanja tijeka rada, uključujući praćenje stoga i relevantne kontekstualne informacije.

Bilježenje se može implementirati korištenjem različitih tehnika, kao što su pisanje u datoteke zapisa, pohranjivanje zapisa u bazu podataka ili slanje zapisa centraliziranoj usluzi za bilježenje. Važno je odabrati okvir za bilježenje koji pruža fleksibilnost, skalabilnost i jednostavnu integraciju s arhitekturom aplikacije.

Evo primjera kako se bilježenje može implementirati u Ruby on Rails aplikaciji korištenjem klase `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 izjava za bilježenje kroz komponente tijeka rada i točke odlučivanja umjetne inteligencije, programeri mogu prikupiti vrijedne informacije za otklanjanje pogrešaka, reviziju i analizu.

Prednosti nadzora i bilježenja

Implementacija nadzora i bilježenja u inteligentnoj orkestraciji tijeka rada nudi nekoliko prednosti:

1. **Otklanjanje pogrešaka i rješavanje problema**: Detaljni zapisi i podaci nadzora pomažu programerima brzo identificirati i dijagnosticirati probleme. Pružaju uvid u tijek izvršavanja rada, interakcije komponenti i sve pogreške ili iznimke na koje se naiđe.

2. **Optimizacija performansi**: Praćenje metrika performansi omogućuje programerima identificiranje uskih grla i optimizaciju komponenti tijeka rada za bolju učinkovitost. Analiziranjem vremena izvršavanja, korištenja resursa i drugih metrika, programeri mogu donositi informirane odluke za poboljšanje ukupnih performansi sustava.

3. **Revizija i usklađenost**: Bilježenje ključnih događaja i odluka pruža revizijski trag za regulatornu usklađenost i odgovornost. Omogućuje organizacijama praćenje i provjeru radnji koje poduzimaju AI komponente te osigurava pridržavanje poslovnih pravila i zakonskih zahtjeva.

4. **Kontinuirano poboljšanje**: Podaci nadzora i bilježenja služe kao vrijedni ulazni podaci za kontinuirano poboljšanje inteligentnih tijekova rada. Analiziranjem povijesnih podataka, identificiranjem obrazaca i mjerenjem učinkovitosti AI odluka, programeri mogu iterativno poboljšavati i unapređivati logiku orkestracije tijeka rada.

Razmatranja i najbolje prakse

Pri implementaciji nadzora i bilježenja u inteligentnoj orkestraciji tijeka rada, razmotrite sljedeće najbolje prakse:

1. **Definiranje jasnih metrika nadzora**: Identificirajte ključne metrike i događaje koje treba pratiti na temelju specifičnih zahtjeva tijeka rada. Usredotočite se na metrike koje pružaju smislene uvide u performanse, zdravlje i ponašanje sustava.

2. **Implementacija detaljnog bilježenja**: Osigurajte da su izjave za bilježenje postavljene na odgovarajućim točkama unutar komponenti tijeka rada i točkama odlučivanja AI-ja. Zabilježite relevantne kontekstualne informacije, poput ulaznih parametara, izlaznih rezultata i svih generiranih međupodataka.

3. **Korištenje strukturiranog bilježenja**: Usvojite format strukturiranog bilježenja kako biste olakšali jednostavno parsiranje i analizu podataka zapisa. Strukturirano bilježenje omogućuje bolje pretraživanje, filtriranje i agregaciju unosa zapisa.

4. **Upravljanje zadržavanjem i rotacijom zapisa**: Implementirajte politike zadržavanja i rotacije zapisa za upravljanje pohranom i životnim ciklusom datoteka zapisa. Odredite odgovarajuće razdoblje zadržavanja na temelju zakonskih zahtjeva, ograničenja pohrane i potreba analize. Ako je moguće, prebacite bilježenje na uslugu treće strane kao što je Papertrail.

5. **Zaštita osjetljivih informacija**: Budite oprezni pri bilježenju osjetljivih informacija, poput osobnih podataka (PII) ili povjerljivih poslovnih podataka. Implementirajte odgovarajuće sigurnosne mjere, poput maskiranja podataka ili enkripcije, kako biste zaštitili osjetljive informacije u datotekama zapisa.

6. **Integracija s alatima za nadzor i upozoravanje**: Iskoristite alate za nadzor i upozoravanje za centralizaciju prikupljanja, analize i vizualizacije podataka nadzora i bilježenja. Ovi alati mogu pružiti uvide u stvarnom vremenu, generirati upozorenja na temelju unaprijed definiranih pragova i olakšati proaktivno otkrivanje i rješavanje problema. Moj omiljeni od ovih alata je Datadog.

Implementacijom sveobuhvatnih mehanizama nadzora i bilježenja, programeri mogu dobiti vrijedne uvide u ponašanje i performanse inteligentnih tijekova rada. Ovi uvidi omogućuju učinkovito otklanjanje pogrešaka, optimizaciju i kontinuirano poboljšanje sustava orkestracije tijeka rada temeljenih na umjetnoj inteligenciji.

Razmatranja skalabilnosti i performansi

Skalabilnost i performanse su ključni aspekti koje treba uzeti u obzir pri dizajniranju i implementaciji sustava inteligentne orkestracije tijeka rada. Kako se povećava volumen istovremenih tijekova rada i složenost komponenti temeljenih na umjetnoj inteligenciji,

postaje ključno osigurati da sustav može učinkovito upravljati radnim opterećenjem i besprijekorno se skalirati kako bi zadovoljio rastuće zahtjeve.

Upravljanje velikim volumenom istovremenih tijekova rada

Sustavi inteligentne orkestracije tijeka rada često trebaju upravljati velikim brojem istovremenih tijekova rada. Za osiguranje skalabilnosti, razmotrite sljedeće strategije:

1. **Asinkrona obrada**: Implementirajte mehanizme asinkrone obrade za odvajanje izvršavanja komponenti tijeka rada. To omogućuje sustavu da upravlja s više tijekova rada istovremeno bez blokiranja ili čekanja da svaka komponenta završi. Asinkrona obrada može se postići korištenjem redova poruka, arhitektura vođenih događajima ili okvira za obradu pozadinskih poslova kao što je Sidekiq.

2. **Distribuirana arhitektura**: Dizajnirajte arhitekturu sustava da koristi serverless komponente (poput AWS Lambda) ili jednostavno distribuirajte radno opterećenje preko više čvorova ili poslužitelja uz vaš glavni aplikacijski poslužitelj. To omogućuje horizontalnu skalabilnost, gdje se mogu dodati dodatni čvorovi za upravljanje povećanim volumenom tijeka rada.

3. **Paralelno izvršavanje**: Identificirajte prilike za paralelno izvršavanje unutar tijekova rada. Neke komponente tijeka rada mogu biti neovisne jedna o drugoj i mogu se izvršavati istovremeno. Korištenjem tehnika paralelne obrade, poput višenitnosti ili distribuiranih redova zadataka, sustav može optimizirati korištenje resursa i smanjiti ukupno vrijeme izvršavanja tijeka rada.

Optimizacija performansi komponenti temeljenih na UI

Komponente temeljene na umjetnoj inteligenciji, poput modela strojnog učenja ili sustava za obradu prirodnog jezika, mogu biti računalno zahtjevne i utjecati na ukupne

performanse sustava za orkestraciju radnih tokova. Za optimizaciju performansi UI komponenti, razmotrite sljedeće tehnike:

1. **Predmemoriranje**: Ako je vaša UI obrada čisto generativna i ne uključuje pretraživanje informacija u stvarnom vremenu ili vanjske integracije za generiranje chatova, možete istražiti mehanizme predmemoriranja za pohranu i ponovnu upotrebu rezultata često pristupanih ili računalno zahtjevnih operacija.

2. **Optimizacija modela**: Kontinuirano optimizirajte način na koji koristite UI modele u komponentama radnog toka. To može uključivati tehnike poput *Destilacije upita* ili jednostavno testiranje novih modela kako postaju dostupni.

3. **Grupna obrada**: Ako radite s modelima klase GPT-4, možda ćete moći iskoristiti tehnike grupne obrade za obradu više podatkovnih točaka ili zahtjeva u jednoj skupini, umjesto njihove pojedinačne obrade. Obradom podataka u grupama, sustav može optimizirati korištenje resursa i smanjiti opterećenje ponavljajućih zahtjeva modela.

Praćenje i profiliranje performansi

Za identificiranje uskih grla u performansama i optimizaciju skalabilnosti inteligentnog sustava za orkestraciju radnih tokova, ključno je implementirati mehanizme praćenja i profiliranja. Razmotrite sljedeće pristupe:

1. **Metrike performansi**: Definirajte i pratite ključne metrike performansi, poput vremena odziva, propusnosti, iskorištenosti resursa i latencije. Ove metrike pružaju uvid u performanse sustava i pomažu identificirati područja za optimizaciju. Popularni agregator UI modela OpenRouter uključuje metrike Poslužitelja[1] i Brzine[2] u svakom API odgovoru, čineći praćenje ovih ključnih metrika trivijalnim.

2. **Alati za profiliranje**: Koristite alate za profiliranje kako biste analizirali performanse pojedinačnih komponenti radnog toka i UI operacija. Alati za profiliranje mogu pomoći

[1] Poslužitelj označava vrijeme potrebno za primanje prvog bajta streamanog generiranja od poslužitelja modela, odnosno "vrijeme do prvog bajta."

[2] Brzina se izračunava kao broj tokena za dovršavanje podijeljen s ukupnim vremenom generiranja. Za zahtjeve koji nisu streaminški, latencija se smatra dijelom vremena generiranja.

u identificiranju kritičnih točaka performansi, neučinkovitih putanja koda ili operacija koje intenzivno koriste resurse. Popularni alati za profiliranje uključuju New Relic, Scout ili ugrađene profilere koje pruža programski jezik ili okvir.

3. **Testiranje opterećenja**: Provodite testiranje opterećenja kako biste procijenili performanse sustava pod različitim razinama istovremenih radnih opterećenja. Testiranje opterećenja pomaže identificirati granice skalabilnosti sustava, otkriti degradaciju performansi i osigurati da sustav može podnijeti očekivani promet bez ugrožavanja performansi.

4. **Kontinuirano praćenje**: Implementirajte mehanizme kontinuiranog praćenja i upozoravanja za proaktivno otkrivanje problema s performansama i uskih grla. Postavite nadzorne ploče i upozorenja za praćenje ključnih pokazatelja performansi (KPI) i primanje obavijesti kada se prekorače unaprijed definirani pragovi. To omogućuje brzu identifikaciju i rješavanje problema s performansama.

Strategije skaliranja

Za upravljanje rastućim radnim opterećenjima i osiguravanje skalabilnosti inteligentnog sustava za orkestraciju radnih tokova, razmotrite sljedeće strategije skaliranja:

1. **Vertikalno skaliranje**: Vertikalno skaliranje uključuje povećanje resursa (npr. CPU, memorija) pojedinačnih čvorova ili poslužitelja za rukovanje većim radnim opterećenjima. Ovaj pristup je prikladan kada sustav zahtijeva više procesorske snage ili memorije za rukovanje složenim radnim tokovima ili UI operacijama.

2. **Horizontalno skaliranje**: Horizontalno skaliranje uključuje dodavanje više čvorova ili poslužitelja sustavu za raspodjelu radnog opterećenja. Ovaj pristup je učinkovit kada sustav treba upravljati velikim brojem istovremenih radnih tokova ili kada se radno opterećenje može lako distribuirati preko više čvorova. Horizontalno skaliranje zahtijeva distribuiranu arhitekturu i mehanizme uravnoteženja opterećenja za osiguravanje ravnomjerne distribucije prometa.

3. **Automatsko skaliranje**: Implementirajte mehanizme automatskog skaliranja za automatsko prilagođavanje broja čvorova ili resursa na temelju potražnje radnog opterećenja. Automatsko skaliranje omogućuje sustavu dinamičko povećavanje ili smanjivanje ovisno o dolaznom prometu, osiguravajući optimalnu iskorištenost resursa i troškovnu učinkovitost. Cloud platforme poput Amazon Web Services (AWS) ili Google Cloud Platform (GCP) pružaju mogućnosti automatskog skaliranja koje se mogu iskoristiti za inteligentne sustave orkestracije radnih tokova.

Tehnike optimizacije performansi

Uz strategije skaliranja, razmotrite sljedeće tehnike optimizacije performansi za poboljšanje učinkovitosti inteligentnog sustava za orkestraciju radnih tokova:

1. **Učinkovita pohrana i dohvat podataka**: Optimizirajte mehanizme pohrane i dohvata podataka koje koriste komponente radnog toka. Koristite učinkovito indeksiranje baze podataka, tehnike optimizacije upita i predmemoriranje podataka kako biste minimizirali latenciju i poboljšali performanse operacija intenzivnih podataka.

2. **Asinkroni U/I**: Koristite asinkrone U/I operacije kako biste spriječili blokiranje i poboljšali odziv sustava. Asinkroni U/I omogućuje sustavu istovremeno rukovanje višestrukim zahtjevima bez čekanja na završetak U/I operacija, čime se maksimizira iskorištenost resursa.

3. **Učinkovita serijalizacija i deserijalizacija**: Optimizirajte procese serijalizacije i deserijalizacije koji se koriste za razmjenu podataka između komponenti tijeka rada. Koristite učinkovite formate serijalizacije, poput Protocol Buffers ili MessagePack, kako biste smanjili opterećenje serijalizacije podataka i poboljšali performanse komunikacije između komponenti.

Za aplikacije temeljene na Rubyju, razmotrite korištenje Universal ID. Universal ID koristi MessagePack i Brotli (kombinaciju izgrađenu za brzinu i najbolju kompresiju podataka u klasi). Kada se kombiniraju, ove biblioteke su do 30% brže i imaju stopu kompresije unutar 2-5% u usporedbi s Protocol Buffers.

4. **Kompresija i kodiranje**: Primijenite tehnike kompresije i kodiranja kako biste smanjili veličinu podataka koji se prenose između komponenti tijeka rada. Algoritmi za kompresiju, poput gzip ili Brotli, mogu značajno smanjiti korištenje mrežne propusnosti i poboljšati ukupne performanse sustava.

Uzimajući u obzir aspekte skalabilnosti i performansi tijekom dizajna i implementacije sustava za inteligentnu orkestraciju tijeka rada, možete osigurati da vaš sustav može upravljati velikim brojem istovremenih tijekova rada, optimizirati performanse komponenti pogonjenih umjetnom inteligencijom i nesmetano se skalirati kako bi zadovoljio rastuće zahtjeve. Kontinuirano praćenje, profiliranje i napori za optimizaciju ključni su za održavanje performansi i odzivnosti sustava kako se radno opterećenje i složenost povećavaju tijekom vremena.

Testiranje i validacija tijekova rada

Testiranje i validacija ključni su aspekti razvoja i održavanja sustava za inteligentnu orkestraciju tijeka rada. S obzirom na složenu prirodu tijekova rada pogonjenih umjetnom inteligencijom, neophodno je osigurati da svaka komponenta funkcionira prema očekivanjima, da se cjelokupni tijek rada ponaša ispravno te da su odluke umjetne inteligencije točne i pouzdane. U ovom odjeljku istražit ćemo različite tehnike i razmatranja za testiranje i validaciju inteligentnih tijekova rada.

Jedinično testiranje komponenti tijeka rada

Jedinično testiranje uključuje testiranje pojedinačnih komponenti tijeka rada u izolaciji kako bi se potvrdila njihova ispravnost i robusnost. Prilikom jediničnog testiranja komponenti pogonjenih umjetnom inteligencijom, razmotrite sljedeće:

1. **Validacija unosa**: Testirajte sposobnost komponente da rukuje različitim vrstama unosa, uključujući valjane i nevaljane podatke. Provjerite upravlja li komponenta elegantno graničnim slučajevima i pruža li odgovarajuće poruke o pogreškama ili iznimke.

2. **Verifikacija izlaza**: Potvrdite da komponenta proizvodi očekivani izlaz za zadani skup ulaznih podataka. Usporedite stvarni izlaz s očekivanim rezultatima kako biste osigurali ispravnost.

3. **Upravljanje pogreškama**: Testirajte mehanizme upravljanja pogreškama komponente simulirajući različite scenarije pogrešaka, poput nevaljanih unosa, nedostupnosti resursa ili neočekivanih iznimki. Provjerite hvata li i upravlja li komponenta pogreškama na odgovarajući način.

4. **Granični uvjeti**: Testirajte ponašanje komponente u graničnim uvjetima, poput praznog unosa, maksimalne veličine unosa ili ekstremnih vrijednosti. Osigurajte da komponenta upravlja ovim uvjetima elegantno bez rušenja ili proizvodnje netočnih rezultata.

Evo primjera jediničnog testa za komponentu tijeka rada u Rubyju koristeći RSpec 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 primjeru, komponenta `OrderValidator` testirana je pomoću dva testna slučaja: jedan za valjanu narudžbu i drugi za nevaljanu narudžbu. Testni slučajevi provjeravaju vraća li metoda `validate` očekivanu boolean vrijednost na temelju valjanosti narudžbe.

Integracijsko testiranje interakcija tijeka rada

Integracijsko testiranje fokusira se na provjeru interakcija i toka podataka između različitih komponenti tijeka rada. Osigurava da komponente besprijekorno rade zajedno i proizvode očekivane rezultate. Prilikom integracijskog testiranja inteligentnih tijekova rada, uzmite u obzir sljedeće:

1. **Interakcija komponenti**: Testirajte komunikaciju i razmjenu podataka između komponenti tijeka rada. Provjerite je li izlaz jedne komponente ispravno proslijeđen kao ulaz sljedećoj komponenti u tijeku rada.

2. **Konzistentnost podataka**: Osigurajte da podaci ostanu konzistentni i točni dok prolaze kroz tijek rada. Provjerite izvode li se transformacije podataka, izračuni i agregacije ispravno.

3. **Propagacija iznimki**: Testirajte kako se iznimke i pogreške propagiraju i obrađuju kroz komponente tijeka rada. Provjerite jesu li iznimke uhvaćene, zapisane i prikladno obrađene kako bi se spriječio prekid tijeka rada.

4**. Asinkrono ponašanje:** Ako tijek rada uključuje asinkrone komponente ili paralelno izvršavanje, testirajte mehanizme koordinacije i sinkronizacije. Osigurajte da se tijek rada ponaša ispravno u konkurentnim i asinkronim scenarijima.

Evo primjera integracijskog testa za tijek rada u Rubyju koristeći RSpec testing framework:

```
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 primjeru, `OrderProcessingWorkflow` se testira provjerom interakcija između različitih komponenti tijeka rada. Testni slučaj postavlja očekivanja za ponašanje svake komponente i osigurava da tijek rada uspješno obrađuje narudžbu, ažurirajući status narudžbe u skladu s tim.

Testiranje AI točaka odlučivanja

Testiranje AI točaka odlučivanja ključno je za osiguravanje točnosti i pouzdanosti tijekova rada koji se temelje na umjetnoj inteligenciji. Prilikom testiranja AI točaka odlučivanja, uzmite u obzir sljedeće:

1. **Točnost odlučivanja**: Provjerite donosi li AI komponenta točne odluke na temelju ulaznih podataka i treniranog modela. Usporedite AI odluke s očekivanim ishodima ili referentnim podacima.

2. **Granični slučajevi**: Testirajte ponašanje AI komponente u graničnim slučajevima i neuobičajenim scenarijima. Provjerite upravlja li AI komponenta ovim slučajevima elegantno i donosi li razumne odluke.

3. **Pristranost i pravednost**: Procijenite AI komponentu na potencijalne pristranosti i osigurajte da donosi pravedne i nepristrane odluke. Testirajte komponentu s raznolikim ulaznim podacima i analizirajte ishode u potrazi za diskriminacijskim obrascima.

4. **Objašnjivost**: Ako AI komponenta pruža objašnjenja ili obrazloženja za svoje odluke, provjerite točnost i jasnoću objašnjenja. Osigurajte da su objašnjenja usklađena s temeljnim procesom donošenja odluka.

Evo primjera testiranja AI točke odlučivanja u Rubyju 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
```

```
14        let(:tx) do
15          build(:transaction, amount: 100, location: 'Low-Risk Country')
16        end
17
18        it 'returns false' do
19          expect(subject.detect_fraud(tx)).to be false
20        end
21      end
22    end
23  end
```

U ovom primjeru, `FraudDetector` AI komponenta testirana je s dva testna slučaja: jedan za prijevarnu transakciju i drugi za legitimnu transakciju. Testni slučajevi provjeravaju vraća li metoda `detect_fraud` očekivanu boolean vrijednost na temelju karakteristika transakcije.

Testiranje s kraja na kraj

Testiranje s kraja na kraj uključuje testiranje cijelog tijeka rada od početka do kraja, simulirajući stvarne scenarije i korisničke interakcije. Osigurava da se tijek rada ponaša ispravno i proizvodi željene rezultate. Prilikom provođenja testiranja s kraja na kraj za inteligentne tijekove rada, uzmite u obzir sljedeće:

1. **Korisnički scenariji**: Identificirajte uobičajene korisničke scenarije i testirajte ponašanje tijeka rada u tim scenarijima. Provjerite obrađuje li tijek rada ispravno korisničke unose, donosi li odgovarajuće odluke i proizvodi li očekivane rezultate.

2. **Validacija podataka**: Osigurajte da tijek rada validira i pročišćava korisničke unose kako bi se spriječile nekonzistentnosti podataka ili sigurnosne ranjivosti. Testirajte tijek rada s različitim vrstama ulaznih podataka, uključujući valjane i nevaljane podatke.

3. **Oporavak od pogrešaka**: Testirajte sposobnost tijeka rada da se oporavi od pogrešaka i iznimki. Simulirajte scenarije pogrešaka i provjerite upravlja li tijek rada njima elegantno, bilježi li pogreške i poduzima li odgovarajuće radnje oporavka.

4. **Performanse i skalabilnost**: Procijenite performanse i skalabilnost tijeka rada pod različitim uvjetima opterećenja. Testirajte tijek rada s velikim brojem istovremenih zahtjeva i mjerite vrijeme odziva, iskorištenost resursa i ukupnu stabilnost sustava.

Evo primjera testa s kraja na kraj za tijek rada u Rubyju koristeći RSpec testni okvir 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 primjeru, testiranje s kraja na kraj simulira korisnika koji postavlja narudžbu putem web sučelja. Popunjava potrebna polja obrasca, šalje narudžbu i provjerava je li narudžba uspješno obrađena, prikazujući odgovarajuću poruku potvrde i ažurirajući status narudžbe u bazi podataka.

Kontinuirana integracija i implementacija

Kako bi se osigurala pouzdanost i održivost inteligentnih tijekova rada, preporučuje se integriranje testiranja i validacije u cjevovod kontinuirane integracije i implementacije (CI/CD). To omogućuje automatizirano testiranje i validaciju promjena u tijeku rada prije njihove implementacije u produkciju. Razmotrite sljedeće prakse:

1. **Automatsko izvršavanje testova**: Konfigurirajte CI/CD cjevovod da automatski pokreće paket testova kad god se naprave promjene u kodu tijeka rada. To osigurava rano otkrivanje regresija ili grešaka u procesu razvoja.

2. **Praćenje pokrivenosti testovima**: Mjerite i pratite pokrivenost testovima komponenti tijeka rada i točaka AI odlučivanja. Težite visokoj pokrivenosti testovima kako biste osigurali temeljito testiranje kritičnih putanja i scenarija.

3. **Kontinuirana povratna informacija**: Integrirajte rezultate testova i metrike kvalitete koda u razvojni tijek rada. Pružite kontinuiranu povratnu informaciju razvijateljima o statusu testova, kvaliteti koda i svim problemima otkrivenim tijekom CI/CD procesa.

4. **Pripremna okruženja**: Implementirajte tijek rada u pripremna okruženja koja vjerno odražavaju produkcijsko okruženje. Provedite dodatno testiranje i validaciju u pripremnom okruženju kako biste otkrili sve probleme vezane uz infrastrukturu, konfiguraciju ili integraciju podataka.

5. **Mehanizmi za povratak**: Implementirajte mehanizme za povratak u slučaju neuspjeha implementacije ili kritičnih problema otkrivenih u produkciji. Osigurajte da se tijek rada može brzo vratiti na prethodnu stabilnu verziju kako bi se minimiziralo vrijeme prekida rada i utjecaj na korisnike.

Uključivanjem testiranja i validacije tijekom cijelog životnog ciklusa razvoja inteligentnih tijekova rada, organizacije mogu osigurati pouzdanost, točnost i održivost svojih sustava temeljenih na umjetnoj inteligenciji. Redovito testiranje i validacija pomažu u otkrivanju grešaka, sprječavanju regresija i izgradnji povjerenja u ponašanje i rezultate tijeka rada.

Dio 2: Obrasci

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Inženjerstvo uputa

Lanac razmišljanja

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Kako funkcionira

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Primjeri

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Generiranje sadržaja

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Stvaranje Strukturiranih Entiteta

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Vođenje LLM agenata

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Prednosti i razmatranja

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Promjena načina rada

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Kako funkcionira

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Kada ga koristiti

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Primjer

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Dodjela uloge

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Kako funkcionira

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Kada ga koristiti

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Primjeri

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Prompt Object

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Kako funkcionira

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Predložak uputa

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Kako funkcionira

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Prednosti i razmatranja

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Kada ga koristiti:

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Primjer

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Strukturirani U/I

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Kako funkcionira

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Skaliranje strukturiranog ulaza/izlaza

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Prednosti i razmatranja

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Ulančavanje promptova

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Kako funkcionira

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Kada ga koristiti

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Primjer: Olympijino uvođenje

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Prepisivač upita

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Kako radi

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Primjer

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Ograđivanje odgovora

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Kako funkcionira

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Prednosti i razmatranja

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Rukovanje pogreškama

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Analizator upita

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Kako funkcionira

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Implementacija

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Označavanje vrsta riječi (POS) i Prepoznavanje imenovanih entiteta (NER)

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Klasifikacija namjere

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Izvlačenje ključnih riječi

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Prednosti

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Query Rewriter

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Kako funkcionira

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Primjer

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Prednosti

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Trbuhozborac

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Kako funkcionira

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Kada ga koristiti

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Primjer

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Diskretne komponente

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Predikat

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Kako funkcionira

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Kada ga koristiti

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Primjer

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

API Fasada

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Kako Funkcionira

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Ključne Prednosti

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Kada Koristiti

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Primjer

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Autentifikacija i Autorizacija

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Obrada Zahtjeva

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Formatiranje Odgovora

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Obrada Pogrešaka i Rubni Slučajevi

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Razmatranja o Skalabilnosti i Performansama

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Usporedba s Drugim Oblikovnim Obrascima

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Interpreter rezultata

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Kako funkcionira

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Kada ga koristiti

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Primjer

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Virtualni stroj

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Kako funkcionira

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Kada ga koristiti

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Primjer

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Iza Čarolije

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Specifikacija i testiranje

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Specificiranje ponašanja

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Pisanje testnih slučajeva

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Primjer: Testiranje komponente za prevođenje

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Ponavljanje HTTP interakcija

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Čovjek u petlji (HITL)

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Visokorizični obrasci

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Hibridna inteligencija

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Adaptivni odziv

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Izmjena uloga između čovjeka i UI-ja

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Eskalacija

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Kako funkcionira

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Ključne prednosti

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Primjena u stvarnom svijetu: Zdravstvo

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Povratna petlja

Kako to funkcionira

Primjene i primjeri

Napredne tehnike u integraciji ljudskih povratnih informacija

Pasivno zračenje informacija

Kako funkcionira

Kontekstualni prikaz informacija

Proaktivne obavijesti

Objašnjavajući uvidi

Interaktivno istraživanje

Ključne prednosti

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Primjene i primjeri

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Suradničko donošenje odluka (CDM)

Kako funkcionira

Primjer

Kontinuirano učenje

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Kako to funkcionira

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Primjene i primjeri

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Primjer

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Etička razmatranja

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Uloga sustava s ljudskom kontrolom u ublažavanju AI rizika

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Tehnološki napredak i budući izgledi

Izazovi i ograničenja HITL sustava

Inteligentno upravljanje pogreškama

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Tradicionalni pristupi upravljanju pogreškama

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Kontekstualna dijagnoza pogreške

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Kako to funkcionira

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Inženjerstvo upita za kontekstualnu dijagnozu pogreške

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Dohvaćanjem potpomognuto generiranje za kontekstualnu dijagnozu pogrešaka

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Inteligentno izvještavanje o pogreškama

Prediktivno sprječavanje pogrešaka

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Kako to funkcionira

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Pametni oporavak od pogrešaka

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Kako to funkcionira

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Personalizirana komunikacija o pogreškama

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Kako funkcionira

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Prilagodljivi tijek rada rukovanja pogreškama

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Kako to funkcionira

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Kontrola kvalitete

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Eval

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Problem

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Rješenje

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Kako funkcionira

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Primjer

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Razmatranja

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Razumijevanje zlatnih referenci

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Kako funkcioniraju evaluacije bez referenci

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Zaštitni mehanizam

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Problem

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Rješenje

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Kako to funkcionira

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Primjer

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Razmatranja

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Zaštitne ograde i evaluacije: Dvije strane istog novčića

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Međusobna zamjenjivost zaštitnih ograda i evaluacija bez referenci

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Implementacija dvojne namjene zaštitnih ograda i evaluacija

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Pojmovnik

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Pojmovnik

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

A

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

B

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

C

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

D

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

E

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

F

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

G

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

H

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

I

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

J

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

K

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

L

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

M

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

N

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

O

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

P

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Q

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

R

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

S

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

T

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

U

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

V

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

W

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Z

Ovaj sadržaj nije dostupan u uzorčnoj knjizi. Knjiga se može kupiti na Leanpubu na http://leanpub.com/patterns-of-application-development-using-ai-hr.

Index