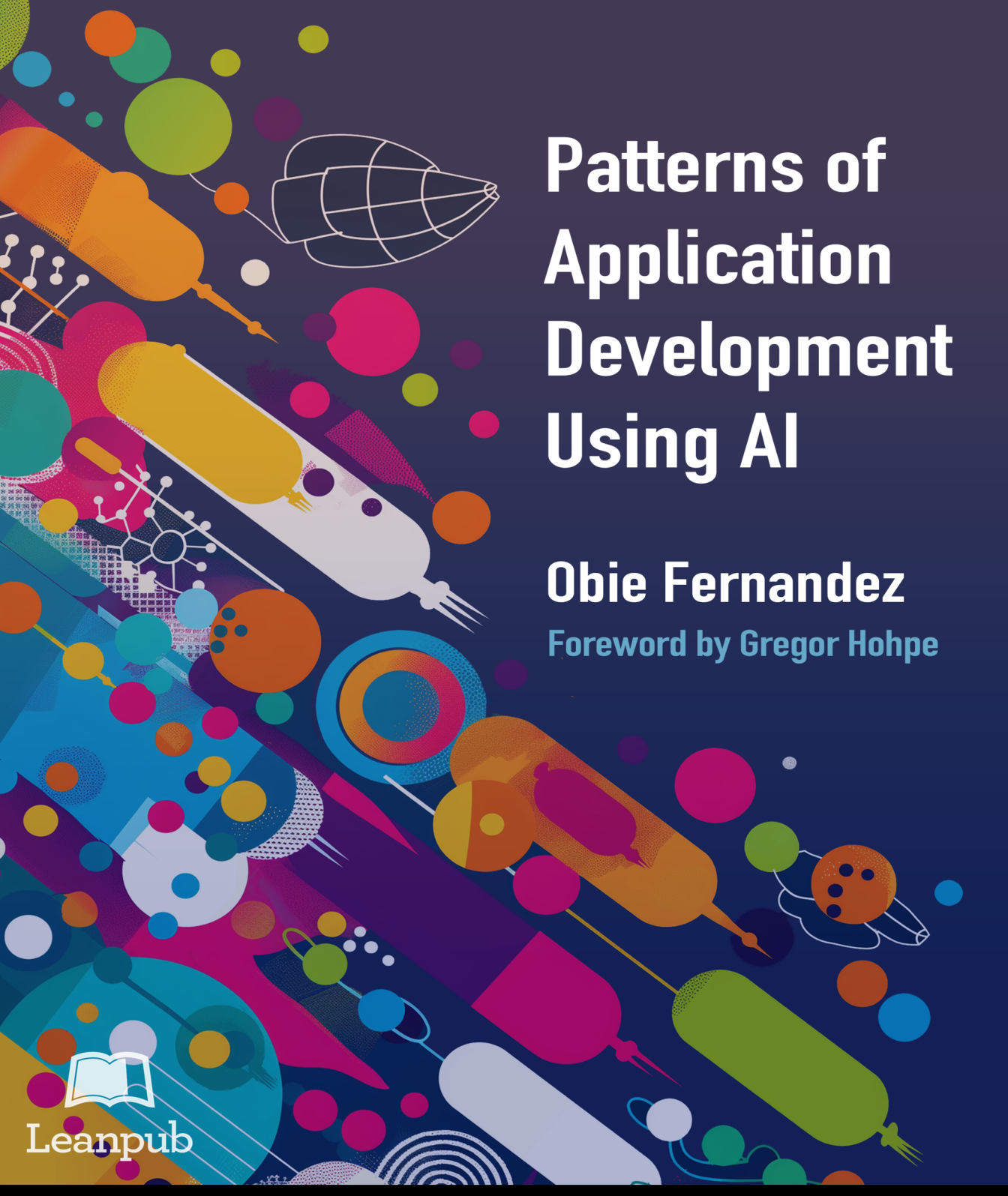


Izdanje na Bosanskom

Obrasci Razvoja Aplikacija Korištenjem UI (Bosnian Edition)

Obie Fernandez

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

Ova verzija je objavljena 2025-01-23

Tweet-ujte Ovu Knjigu!

Molimo pomožite Obie Fernandez širenjem riječi o ovoj knjizi na Twitteru!

Predloženi hashtag za ovu knjigu je #poaduai.

Saznajte što drugi ljudi govore o knjizi klikom na ovaj link za pretragu ovog hashtaga na Twitteru:

#poaduai

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

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

Contents

Predgovor autora Gregor Hohpe

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Predgovor

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

O knjizi

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

O primjerima koda

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Šta ne obrađujem

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Za koga je ova knjiga

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Izgradnja zajedničkog vokabulara

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Uključivanje

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Zahvale

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Šta je sa ilustracijama?

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

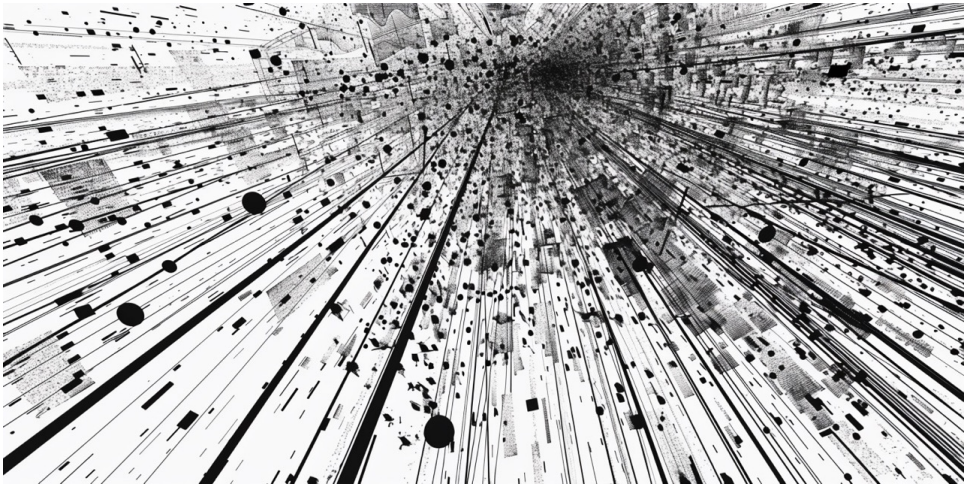
O Lean Publishingu

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

O Autoru

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Uvod



Ako ste željni da počnete integrirati velike jezičke modele (VJM) u svoje programerske projekte, slobodno se udubite direktno u obrasce i primjere koda predstavljene u kasnijim poglavljima. Međutim, da biste u potpunosti cijenili snagu i potencijal ovih obrazaca, vrijedi odvojiti trenutak da razumijete širi kontekst i kohezivni pristup koji oni predstavljaju.

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

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

između ovih obrazaca, možete donositi informisane odluke o tome gdje i kako najefikasnije primijeniti vještačku inteligenciju.

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

Razmišljanja o softverskoj arhitekturi

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

Jedan od razloga zbog kojeg sam lično toliko uzbuđen zbog promjena koje donosi ovaj moćni novi val AI tehnologije je njegov uticaj na ono što smatramo odlukama *softverske arhitekture*. On izaziva tradicionalne pojmove o tome šta čini "ispravan" način dizajniranja i implementacije naših softverskih projekata. Također dovodi u pitanje može li se arhitektura i dalje smatrati prvenstveno *onim dijelovima sistema koje je teško promijeniti*, s obzirom da AI unapređenje čini lakšim nego ikad promjenu bilo kojeg dijela vašeg projekta, u bilo koje vrijeme.

Možda ulazimo u vrhunske godine "postmodernog" pristupa softverskom inženjerstvu. U ovom kontekstu, postmoderno se odnosi na fundamentalni pomak od tradicionalnih paradigmi, gdje su programeri bili odgovorni za pisanje i održavanje svake linije koda. Umjesto toga, prihvata se ideja delegiranja zadataka, kao što su manipulacija podacima, složeni algoritmi, pa čak i čitavi dijelovi aplikacijske logike, bibliotekama trećih strana i eksternim API-jima. Ovaj postmoderni pomak predstavlja značajno odstupanje 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 neophodno napisati, na osnovu učenja Larry Wall-a i drugih hakerskih luminara poput njega. Minimiziranjem količine napisanog koda, možemo se kretati brže, smanjiti površinu za bugove, pojednostaviti održavanje i poboljšati ukupnu pouzdanost naših aplikacija. Manje koda nam omogućava da se fokusiramo na osnovnu poslovnu logiku i korisničko iskustvo, dok delegiramo drugi posao drugim servisima.

Sada kada AI sistemi mogu obavljati zadatke koji su ranije bili ekskluzivni domen koda koji pišu ljudi, trebali bismo biti još produktivniji i agilniji, s većim fokusom nego ikad na stvaranju poslovne vrijednosti i korisničkog iskustva.

Naravno, postoje kompromisi delegiranja ogromnih dijelova vašeg projekta AI sistemima, kao što su potencijalni gubitak kontrole i potreba za robusnim mehanizmima praćenja i povratnih informacija. Zato to zahtijeva novi set vještina i znanja, uključujući barem neko osnovno razumijevanje kako AI funkcioniše.

Šta je veliki jezički model?

Veliki jezički modeli (VJM) su vrsta modela vještačke inteligencije koji su privukli značajnu pažnju posljednjih godina, još od lansiranja GPT-3 od strane OpenAI 2020. godine. VJM-ovi su dizajnirani da obrađuju, razumiju i generišu ljudski jezik s izuzetnom preciznošću i tečnošću. U ovom odjeljku, kratko ćemo pogledati kako VJM-ovi rade i zašto su pogodni za izgradnju inteligentnih sistemskih komponenti.

U svojoj srži, VJM-ovi su zasnovani na algoritmima dubokog učenja, posebno neuralnim mrežama. Ove mreže su sastavljene od međusobno povezanih čvorova, ili neurona, koji obrađuju i prenose informacije. Arhitektura izbora za VJM-ove je često Transformer model, koji se pokazao vrlo efikasnim u obradi sekvencijalnih podataka poput teksta.

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

da efikasnije uhvate zavisnosti na većim udaljenostima. Oni imaju slojeve mehanizama pažnje koji pomažu modelu da se fokusira na različite dijelove ulaznih podataka kako bi razumio kontekst i odnose.

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

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

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

Nakon treniranja, veliki jezički modeli se mogu koristiti za širok spektar jezičkih zadataka. Oni mogu generisati tekst nalik ljudskom, odgovarati na pitanja, sumirati dokumente, prevoditi jezike, pa čak i pisati kod. Svestranost velikih jezičkih modela čini ih vrijednim za izgradnju inteligentnih sistemskih komponenti koje mogu komunicirati s korisnicima, obrađivati i analizirati tekstualne podatke te generisati smislene izlaze.

Uključivanjem velikih jezičkih modela u arhitekturu aplikacije, možete stvoriti AI komponente koje razumiju i obrađuju korisnički unos, generišu dinamički sadržaj i pružaju inteligentne preporuke ili akcije. Ali rad s velikim jezičkim modelima zahtijeva

pažljivo razmatranje resursnih zahtjeva i kompromisa performansi. Veliki jezički modeli su računski intenzivni i mogu zahtijevati značajnu procesorsku snagu i memoriju (drugim riječima, novac) za rad. Većina nas će morati procijeniti troškovne implikacije integracije velikih jezičkih modela u naše aplikacije i djelovati u skladu s tim.

Razumijevanje zaključivanja

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

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

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

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

Veliki jezički modeli dolaze u mnogim veličinama i varijantama

Iako su praktično svi popularni veliki jezički modeli zasnovani na istoj osnovnoj transformer arhitekturi i trenirani na ogromnim tekstualnim skupovima podataka,

dolaze u različitim veličinama i fino su podešeni za različite svrhe. Veličina velikog jezičkog modela, mjerena brojem parametara u njegovoj neuronskoj mreži, ima veliki uticaj na njegove sposobnosti. Veći modeli s više parametara, poput GPT-4, za koji se govori da ima 1 do 2 triliona parametara, općenito su više informisani i sposobniji od manjih modela. Međutim, veći modeli također zahtijevaju mnogo više računarske snage za rad, što se prevodi u veći trošak kada ih koristite putem API poziva.

Da bi veliki jezički modeli bili praktičniji i prilagođeni specifičnim slučajevima upotrebe, osnovni modeli se često fino podešavaju na ciljanijim skupovima podataka. Na primjer, veliki jezički model može biti treniran na velikom korpusu dijaloga da bi se specijalizirao za konverzacijsku AI. Drugi su trenirani na kodu da bi im se usadilo programersko znanje. Postoje čak i modeli koji su posebno trenirani za interakcije u stilu igranja uloga s korisnicima!

Modeli pretraživanja naspram generativnih modela

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

Modeli zasnovani na pretraživanju

Modeli zasnovani na pretraživanju, poznati i kao modeli za pretraživanje informacija, generišu odgovore pretražujući veliku bazu postojećeg teksta i birajući najrelevantnije odlomke na osnovu ulaznog upita. Ovi modeli ne generišu novi tekst iz početka, već spajaju dijelove iz baze podataka kako bi formirali koherentan odgovor.

Jedna od glavnih prednosti modela zasnovanih na pretraživanju je njihova sposobnost da pruže činjenično tačne i ažurne informacije. Budući da se oslanjaju na bazu kuriranog teksta, mogu izvući relevantne informacije iz pouzdanih izvora i predstaviti ih korisniku.

Ovo ih čini pogodnim za aplikacije koje zahtijevaju precizne, činjenične odgovore, kao što su sistemi za odgovaranje na pitanja ili baze znanja.

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

U ovoj knjizi ne obrađujemo upotrebu čistih modela pretraživanja.

Generativni modeli

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

Glavna snaga generativnih modela je njihova sposobnost da proizvedu kreativan, koherentan i kontekstualno relevantan tekst. Mogu voditi otvorene razgovore, generisati priče, pa čak i pisati kod. Ovo ih čini idealnim za aplikacije koje zahtijevaju otvorenije i dinamičnije interakcije, kao što su chatbotovi, kreiranje sadržaja i asistenti za kreativno pisanje.

Međutim, generativni modeli ponekad mogu proizvesti nedosljedne ili činjenično netačne informacije, jer se oslanjaju na obrasce naučene tokom treninga umjesto na kuriranu bazu činjenica. Također mogu biti skloniji pristrasnostima i halucinacijama, generišući 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 kombinuje i pristup pretraživanja i generativni pristup u hibridnom modelu. Ovi modeli koriste tehnike pretraživanja da

pronađu relevantne informacije iz baze podataka, a zatim koriste generativne tehnike da sintetizuju te informacije u koherentan odgovor.

Hibridni modeli nastoje kombinovati činjeničnu tačnost modela zasnovanih na pretraživanju sa sposobnostima generisanja prirodnog jezika generativnih modela. Oni mogu pružiti pouzdanije i ažurnije informacije dok istovremeno zadržavaju sposobnost vođenja otvorenih razgovora.

Pri odabiru između modela zasnovanih na pretraživanju i generativnih modela, trebali biste razmotriti specifične zahtjeve vaše aplikacije. Ako je primarni cilj pružanje tačnih, činjeničnih informacija, model zasnovan na pretraživanju 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 tačnost i generisanje prirodnog jezika.

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

Podešavanje instrukcija

Podešavanje instrukcija je podskup konverzacijskog treninga koji koristi ljudski napisane upite i odgovore kako bi LLM-ovi bolje razumjeli i slijedili instrukcije. Evo nekoliko jednostavnih primjera koji ilustruju kako podešavanje instrukcija na ljudski napisanim upitima i odgovorima čini LLM-ove boljim u razumijevanju i praćenju instrukcija:

Primjer 1: Korisnička podrška

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

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

Ljudski napisan odgovor: "Žao mi je što proizvod nije ispunio vaša očekivanja. Da bismo obradili vaš povrat novca, molimo vas da navedete broj narudžbe i razlog za povrat. Pregledaćemo vaš zahtjev i javiti vam se u roku od 2-3 radna dana."

Primjer 2: Obrazovna pomoć

Instrukcija: "Objasnite koncept fotosinteze desetogodišnjaku."

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

Ljudski napisani odgovor: "Naravno! Biljke prave svoju hranu kroz proces koji se zove fotosinteza. Koriste sunčevu svjetlost, vodu i ugljen-dioksid da stvore hranu i kiseonik. Lišće biljke ima poseban dio koji se zove hlorofil koji im pomaže da uhvate sunčevu svjetlost."

Primjer 3: Lični asistent

Instrukcija: "Zakaži sastanak za sljedeći utorak u 15:00."

> Ljudski napisani upit: “Možeš li zakazati sastanak za utorak u 15:00?”
>
> Ljudski napisani odgovor: “Naravno, zakazao sam vam sastanak za sljedeći utorak u 15:00. Treba li vam još nešto?”

Rezultat je raznolik ekosistem velikih jezičkih modela različitih veličina i sa različitim specijalizacijama. Manji modeli u rasponu od 1-7 milijardi parametara pružaju dobre opšte jezičke sposobnosti dok su efikasniji za pokretanje.

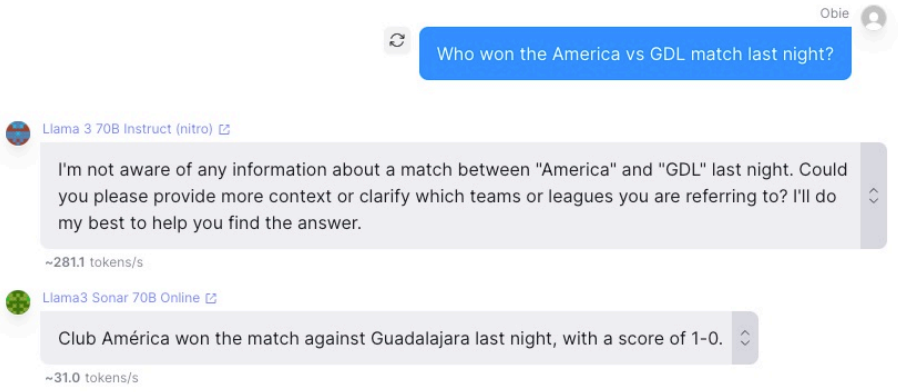
- Mistral 7B
- Llama 3 8B
- Gemma 7B

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

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

Prilikom odabira velikog jezičkog modela za integraciju u aplikaciju, morate uravnotežiti sposobnosti modela sa praktičnim faktorima kao što su troškovi, latencija, dužina konteksta i filtriranje sadržaja. Manji modeli, podešeni za instrukcije, često su najbolji izbor za jednostavnije jezičke zadatke, dok najveći modeli mogu biti potrebni za složeno rezonovanje ili analizu. Podaci za treniranje modela su također važno razmatranje, jer određuju datum do kojeg seže znanje modela.

Određeni modeli, poput nekih od Perplexity-ja, povezani su sa izvorima informacija u realnom vremenu, tako da efektivno nemaju datum prekida. Kada im postavite pitanja, mogu samostalno odlučiti da izvrše pretraživanje weba i dohvate proizvoljne web stranice kako bi generisali odgovor.

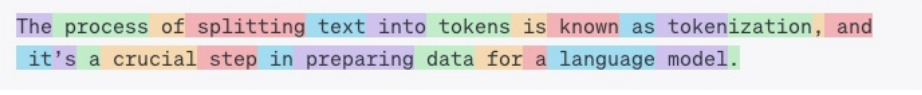


Slika 1. Llama3 sa i bez online pristupa

Na kraju, ne postoji univerzalni veliki jezički model. Razumijevanje varijacija u veličini modela, arhitekturi i treniranju ključno je za odabir pravog modela za dati slučaj upotrebe. Eksperimentisanje s različitim modelima je jedini praktični način da se otkrije koji modeli pružaju najbolje performanse za zadati zadatak.

Tokenizacija: Razbijanje teksta na dijelove

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



Slika 2. Ova rečenica sadrži 27 tokena

Različiti veliki jezički modeli koriste različite strategije tokenizacije, što može imati značajan uticaj na performanse i mogućnosti modela. Neki uobičajeni tokenizatori koje

koriste veliki jezički modeli uključuju:

- **GPT (Kodiranje parova bajtova)**: GPT tokenizatori koriste tehniku koja se zove kodiranje parova bajtova (BPE) za razbijanje teksta na podriječne jedinice. BPE iterativno spaja najčešće parove bajtova u tekstualnom korpusu, formirajući rječnik podriječnih tokena. Ovo omogućava tokenizatoru da obradi rijetke i nove riječi razbijajući ih na češće podriječne dijelove. GPT tokenizatori se koriste u modelima kao što su GPT-3 i GPT-4.

- **Llama (SentencePiece)**: Llama tokenizatori koriste SentencePiece biblioteku, koja je samostalni tokenizator i detokenizator teksta. SentencePiece tretira ulazni tekst kao niz Unicode znakova i uči vokabular podriječi na osnovu korpusa za treniranje. Može obrađivati bilo koji jezik koji se može kodirati u Unicode-u, što ga čini pogodnim za višejezične modele. Llama tokenizatori se koriste u modelima kao što su Meta-in Llama i Alpaca.

- **SentencePiece (Unigram)**: SentencePiece tokenizatori također mogu koristiti drugačiji algoritam pod nazivom Unigram, koji se zasniva na tehnici regularizacije podriječi. Unigram tokenizacija određuje optimalni vokabular podriječi na osnovu unigram jezičkog modela, koji dodjeljuje vjerovatnoće pojedinačnim jedinicama podriječi. Ovaj pristup može proizvesti semantički značajnije podriječi u poređenju sa BPE. SentencePiece sa Unigramom koriste modeli poput Google-ovog T5 i BERT-a.

- **Google Gemini (Multimodalna Tokenizacija)**: Google Gemini koristi shemu tokenizacije dizajniranu za obradu različitih vrsta podataka, uključujući tekst, slike, audio, video i kod. Ova multimodalna sposobnost omogućava Geminiju da obrađuje i integriše različite oblike informacija. Posebno je značajno da Google Gemini 1.5 Pro ima kontekstni prozor koji može obrađivati milione tokena, što je mnogo više nego prethodni modeli. Ovaj opsežni kontekstni prozor omogućava

modelu da obradi veći kontekst, potencijalno vodeći 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 na korištenje modela poput GPT-a, jer se Google-ovo određivanje cijena zasniva na znakovima umjesto na tokenima.

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

- **Veličina vokabulara**: Tokenizator određuje veličinu vokabulara modela, koji predstavlja skup jedinstvenih tokena koje prepoznaje. Veći, detaljniji vokabular može pomoći modelu da obradi širi spektar riječi i fraza i čak postane multimodalan (sposoban za razumijevanje i generisanje više od samog teksta), ali također povećava memorijske zahtjeve modela i računsku složenost.
- **Rukovanje rijetkim i nepoznatim riječima**: Tokenizatori koji koriste jedinice podriječi, poput BPE i SentencePiece-a, mogu razbiti rijetke i nepoznate riječi na češće dijelove podriječi. Ovo omogućava modelu da napravi obrazovane pretpostavke o značenju riječi koje nije ranije vidio, na osnovu podriječi koje sadrže.
- **Višejezična podrška**: Tokenizatori poput SentencePiece-a, koji mogu obrađivati bilo koji jezik koji se može kodirati u Unicode-u, posebno su pogodni 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 uklapa u specifične potrebe obrade jezika za dati zadatak. Tokenizator može imati značajan uticaj na sposobnost modela da obrađuje terminologiju specifičnu za određenu oblast, rijetke riječi i višejezični tekst.

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

Kada se govori o jezičkim modelima, veličina konteksta se odnosi na količinu teksta koju model može razmotriti prilikom obrade ili generisanja svojih odgovora. To je u suštini mjera koliko informacija model može "zapamtiti" i koristiti za oblikovanje svojih izlaza (izraženo u tokenima). Veličina konteksta jezičkog modela može imati značajan uticaj na njegove mogućnosti i vrste zadataka koje može efikasno izvršavati.

Šta je Veličina Konteksta?

U tehničkom smislu, veličina konteksta je određena brojem tokena (riječi ili dijelova riječi) koje jezički model može obraditi u jednom ulaznom nizu. Ovo se često naziva "raspon pažnje" ili "kontekstni prozor" modela. Što je veća veličina konteksta, to više teksta model može razmotriti odjednom pri generisanju odgovora ili izvršavanju zadatka.

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

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

Zašto je veličina konteksta važna?

Veličina konteksta jezičkog modela ima značajan uticaj na njegovu sposobnost razumijevanja i generisanja koherentnog, kontekstualno relevantnog teksta. Evo nekoliko ključnih razloga zašto je veličina konteksta bitna:

1. **Razumijevanje dugačkih sadržaja**: Modeli s većom veličinom konteksta mogu bolje razumjeti i analizirati duže tekstove, kao što su članci, izvještaji, pa čak i cijele knjige. Ovo je ključno za zadatke poput sažimanja dokumenata, odgovaranja na pitanja i analize sadržaja.

2. **Održavanje koherentnosti**: Veći kontekstni prozor omogućava modelu da održi koherentnost i dosljednost kroz duže dijelove izlaznog teksta. Ovo je važno za zadatke poput generisanja priča, sistema za dijalog i kreiranje sadržaja, gdje je održavanje dosljedne naracije ili teme od suštinske važnosti. Također je apsolutno ključno kada se VJM-ovi koriste za generisanje ili transformaciju strukturiranih podataka.
3. **Hvatanje zavisnosti dugog dometa**: Neki jezički zadaci zahtijevaju razumijevanje odnosa između riječi ili fraza koje su udaljene u tekstu. Modeli s većom veličinom konteksta su bolje opremljeni za hvatanje ovih zavisnosti dugog dometa, što može biti važno za zadatke poput analize sentimenta, prevođenja i razumijevanja jezika.
4. **Rukovanje složenim uputstvima**: U aplikacijama gdje se jezički modeli koriste za praćenje složenih, višekoračnih uputstava, veća veličina konteksta omogućava modelu da razmotri cijeli set uputstava pri generisanju odgovora, umjesto samo nekoliko najnovijih riječi.

Primjeri jezičkih modela s različitim veličinama konteksta

Evo nekoliko primjera jezičkih 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-jevog GPT-4, nude različite varijante s većim veličinama konteksta (npr. "32k" verzije), koje mogu rukovati još dužim ulaznim sekvencama. A trenutno (april 2024.) Google Gemini Pro se hvali sa skoro 3 miliona tokena!

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

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

Odabir prave veličine konteksta

Pri odabiru jezičkog modela za određenu aplikaciju, važno je razmotriti zahtjeve zadatka u pogledu veličine konteksta. Za zadatke koji uključuju kratke, izolovane dijelove teksta, poput analize sentimenta ili jednostavnog odgovaranja na pitanja, manja veličina konteksta može biti dovoljna. Međutim, za zadatke koji zahtijevaju razumijevanje i generisanje dužih, složenijih tekstova, veća veličina konteksta će vjerovatno biti neophodna.

Vrijedi napomenuti da veće veličine konteksta često dolaze s povećanim računarskim troškovima i sporijim vremenom obrade, jer model mora razmotriti više informacija pri

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

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

Pronalaženje Igala u Plastu Sijena

Koncept pronalaženja igle u plastu sijena dugo je bio metafora za izazove pretraživanja u velikim skupovima podataka. U domenu VJM-ova (velikih jezičkih 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 svuda. Ovaj scenarij više liči na pronalaženje nekoliko igala u prostranom polju, ne samo u jednom plastu sijena. Evo začkoljice: ne samo da moramo locirati te igle, već ih moramo i uplesti u koherentnu nit.

Kada se suoče sa zadatkom pronalaženja i rasuđivanja o višestrukim činjenicama ugrađenim u dugačke kontekste, VJM-ovi se suočavaju s dvostrukim izazovom. Prvo, tu je jednostavan problem tačnosti pretraživanja—ona prirodno opada kako se broj činjenica povećava. Ovo je očekivano; na kraju krajeva, praćenje višestrukih detalja kroz obiman tekst opterećuje čak i najsofisticiranije modele.

Drugo, i možda kritičnije, je izazov rasuđivanja s tim činjenicama. Jedno je izdvojiti činjenice; sasvim drugo je sintetizirati ih u koherentnu priču ili odgovor. Tu dolazi pravi test. Performanse VJM-ova u zadacima rasuđivanja imaju tendenciju da se pogoršavaju

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

Zašto se ovo događa? Pa, razmotrite dinamiku pamćenja i pažnje u ljudskoj kogniciji, koja se do određene mjere ogleda u VJM-ovima. Prilikom obrade velikih količina informacija, VJM-ovi, poput ljudi, mogu izgubiti trag ranijih detalja dok upijaju nove. Ovo je posebno tačno kod modela koji nisu eksplicitno dizajnirani da automatski prioritiziraju ili se vraćaju na ranije segmente teksta.

Štaviše, sposobnost VJM-a da uplete ove pronađene činjenice u koherentan odgovor slična je izgradnji narativa. Ovo ne zahtijeva samo pronalaženje informacija već duboko razumijevanje i kontekstualno postavljanje, što ostaje težak izazov za trenutnu umjetnu inteligenciju.

Dakle, šta ovo znači za nas kao razvijatelje i integratore ovih tehnologija? Moramo biti izuzetno svjesni ovih ograničenja kada dizajniramo sisteme koji se oslanjaju na VJM-ove za rukovanje složenim zadacima s dugačkim tekstovima. Razumijevanje da se performanse mogu pogoršati pod određenim uslovima pomaže nam da postavimo realna očekivanja i konstruišemo bolje rezervne mehanizme ili dopunske strategije.

Modaliteti: Izvan Teksta

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

Šta su Modaliteti?

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

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

- **Slike**: Vizualni podaci poput fotografija, ilustracija i dijagrama.
- **Audio**: Zvučni podaci poput govora, muzike i zvukova iz okoline.
- **Video**: Pokretni vizualni podaci, često praćeni zvukom, poput video klipova i filmova.

Svaki modalitet predstavlja jedinstvene izazove i mogućnosti za jezičke 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 Jezički Modeli

Multimodalni jezički modeli su dizajnirani da rukuju s više modaliteta unutar jednog modela. Ovi modeli tipično imaju specijalizirane komponente ili slojeve koji mogu i razumjeti unose i generisati izlazne podatke u različitim modalitetima. Neki značajni primjeri multimodalnih jezičkih modela uključuju:

- **OpenAI-jev GPT-4o**: GPT-4o je veliki jezički model koji prirodno razumije i obrađuje govorni audio pored teksta. Ova sposobnost omogućava GPT-4o da izvršava zadatke poput transkripcije govornog jezika, generisanja teksta iz audio unosa i pružanja odgovora na osnovu govornih upita.
- **OpenAI-jev GPT-4 sa vizualnim unosom**: GPT-4 je veliki jezički model koji može obrađivati i tekst i slike. Kada dobije sliku kao unos, GPT-4 može analizirati sadržaj slike i generisati tekst koji opisuje ili odgovara na vizualne informacije.
- **Google-ov Gemini**: Gemini je multimodalni model koji može rukovati tekstom, slikama i videom. Koristi jedinstvenu arhitekturu koja omogućava međumodalno razumijevanje i generisanje, omogućavajući zadatke poput opisivanja slika, sažimanja videa i vizualnog odgovaranja na pitanja.

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

Prednosti i primjene multimodalnih modela

Multimodalni jezički modeli nude nekoliko prednosti i omogućavaju širok spektar primjena, uključujući:

- **Poboljšano razumijevanje**: Obradom informacija iz više modaliteta, ovi modeli mogu steći sveobuhvatnije razumijevanje svijeta, slično načinu na koji ljudi uče iz različitih čulnih inputa.
- **Krosmodalna generacija**: Multimodalni modeli mogu generisati sadržaj u jednom modalitetu na osnovu unosa iz drugog, kao što je kreiranje slike iz tekstualnog opisa ili generisanje video sažetka iz pisanog članka.
- **Pristupačnost**: Multimodalni modeli mogu učiniti informacije pristupačnijim prevođenjem između modaliteta, kao što je generisanje tekstualnih opisa slika za korisnike sa oštećenim vidom ili kreiranje audio verzija pisanog sadržaja.
- **Kreativne primjene**: Multimodalni modeli se mogu koristiti za kreativne zadatke poput generisanja umjetnosti, muzike ili videa na osnovu tekstualnih promptova, otvarajući nove mogućnosti za umjetnike i kreatore sadržaja.

Kako multimodalni jezički modeli nastavljaju napredovati, vjerovatno će igrati sve važniju ulogu u razvoju aplikacija pogonjenih vještačkom inteligencijom koje mogu razumjeti i generisati sadržaj kroz više modaliteta. Ovo će omogućiti prirodnije i intuitivnije interakcije između ljudi i AI sistema, kao i otključati nove mogućnosti za kreativno izražavanje i širenje znanja.

Ekosistemi pružalaca usluga

Kada je riječ o uključivanju velikih jezičkih modela (LLM-ova) u aplikacije, imate rastući izbor opcija. Svaki veliki pružalac LLM-ova, kao što su OpenAI, Anthropic, Google i Cohere, nudi svoj vlastiti ekosistem modela, API-ja i alata. Izbor pravog pružaoca uključuje razmatranje različitih faktora, uključujući cijene, performanse, filtriranje sadržaja, privatnost podataka i opcije prilagođavanja.

OpenAI

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

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

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

Anthropic

Anthropic je još jedan veliki igrač u LLM prostoru, sa njihovim Claude modelima koji stiču popularnost zbog snažnih performansi i etičkih razmatranja. Anthropic se fokusira na razvoj sigurnih i odgovornih AI sistema, sa snažnim naglaskom na filtriranje sadržaja i izbjegavanje štetnih izlaza.

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

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

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

Google

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

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

Meta

Meta, ranije poznata kao Facebook, duboko je uključena u razvoj velikih jezičkih modela, što je istaknuto njihovim izdavanjem modela poput LLaMA i OPT. Ovi modeli se ističu po svojim snažnim performansama u različitim jezičkim zadacima i uglavnom su dostupni kroz kanale otvorenog koda, podržavajući Meta-inu predanost istraživanju i saradnji sa zajednicom.

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

Pored svojih tehničkih ponuda, Meta stavlja snažan naglasak na etički razvoj AI-ja. Oni implementiraju robustno filtriranje sadržaja i fokusiraju se na smanjenje pristrasnosti, u skladu sa svojim širim ciljevima sigurnosti i odgovornosti u AI aplikacijama.

Cohere

Cohere je noviji učesnik u području LLM-a, fokusirajući se na to da LLM-ovi budu pristupačniji i lakši za korištenje od konkurencije. Njihov ekosistem uključuje Cohere API, koji pruža pristup nizu unaprijed treniranih modela za zadatke poput generisanja teksta, klasifikacije i sumiranja.

Cohere također nudi alate za inženjering promptova, fino podešavanje i filtriranje sadržaja. Oni naglašavaju privatnost i sigurnost podataka, sa funkcijama poput šifriranog skladištenja podataka i kontrole pristupa.

Ollama

Ollama je samostalno hostovana platforma koja korisnicima omogućava upravljanje i implementaciju različitih velikih jezičkih modela (LLM-ova) lokalno na njihovim mašinama, dajući im potpunu kontrolu nad njihovim AI modelima bez oslanjanja na eksterne cloud usluge. Ovo podešavanje je idealno za one koji prioritiziraju privatnost podataka i žele da upravljaju svojim AI operacijama interno.

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

Za programere koji žele 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 promptove, omogućavajući visok stepen prilagođavanja za prilagođavanje modela specifičnim potrebama ili slučajevima upotrebe.

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

modela. Ovo je čini moćnim alatom za firme i pojedince koji zahtijevaju robustne AI mogućnosti bez kompromitovanja sigurnosti i kontrole.

Multi-Model Platforme

Dodatno, postoje pružaoci usluga koji hostuju širok spektar modela otvorenog koda, kao što su Together.ai i Groq. Ove platforme nude fleksibilnost i prilagodljivost, omogućavajući vam da pokrenete i, u nekim slučajevima, čak fino podesite modele otvorenog koda prema vašim specifičnim potrebama. Na primjer, Together.ai pruža pristup nizu LLM-ova otvorenog koda, omogućavajući korisnicima da eksperimentišu sa različitim modelima i konfiguracijama. Groq se fokusira na pružanje ultra visokih performansi koje u vrijeme pisanja ove knjige djeluju gotovo magično

Odabir LLM Pružaoca Usluga

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

- **Cijena**: Različiti pružaoci usluga nude različite modele cijena, od plaćanja po korištenju do pretplatničkih planova. Važno je razmotriti očekivanu upotrebu i budžet pri odabiru pružaoca usluga.
- **Performanse**: Performanse LLM-ova mogu značajno varirati između pružalaca usluga, pa je važno testirati modele na specifičnim slučajevima upotrebe prije donošenja odluke.
- **Filtriranje sadržaja**: Zavisno od aplikacije, filtriranje sadržaja može biti kritično razmatranje. Neki pružaoci usluga nude robusnije opcije filtriranja sadržaja od drugih.
- **Privatnost podataka**: Ako aplikacija rukuje osjetljivim korisničkim podacima, važno je odabrati pružaoca usluga sa snažnim praksama privatnosti i sigurnosti podataka.
- **Prilagođavanje**: Neki pružaoci usluga nude više fleksibilnosti u pogledu finog podešavanja i prilagođavanja modela za specifične slučajeve upotrebe.

U konačnici, izbor LLM pružaoca usluga zavisi od specifičnih zahtjeva i ograničenja aplikacije. Pažljivim procjenjivanjem opcija i razmatranjem faktora poput cijena, performansi i privatnosti podataka, možete odabrati pružaoca usluga koji najbolje odgovara vašim potrebama.

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

OpenRouter

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

Razmišljanje o performansama

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

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

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

Nekoliko faktora može uticati na performanse jezičkog modela, uključujući:

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

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

Jedna od lijepih stvari kod OpenRouter-a je da za mnoge modele koje nudi dobijate izbor cloud provajdera s različitim profilima performansi i troškova.

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

Grupna obrada: Obrada više zahtjeva istovremeno u grupama može poboljšati propusnost amortizacijom režijskih troškova učitavanja modela i prenosa podataka.

Keširanje: Keširanje rezultata često korištenih promptova ili ulaznih sekvenci može smanjiti broj zahtjeva za zaključivanje i poboljšati ukupne performanse.

Pri odabiru jezičkog modela za produkcijsku aplikaciju, važno je testirati njegove performanse na reprezentativnim radnim opterećenjima i hardverskim konfiguracijama. Ovo može pomoći u identifikaciji potencijalnih uskih grla i osigurati da model može ispuniti zahtijevane 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 realnom vremenu, dok veći, snažniji model može biti bolje prilagođen za grupnu obradu ili složene zadatke rezonovanja.

Eksperimentisanje s različitim LLM modelima

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

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

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

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

Složeni AI sistemi

Prije zatvaranja našeg uvoda, vrijedi spomenuti da su prije 2023. godine i eksplozije interesa za generativnu AI potaknutu ChatGPT-om, tradicionalni AI pristupi obično se oslanjali na integraciju pojedinačnih, zatvorenih modela. Nasuprot tome,

Složeni AI sistemi koriste kompleksne cjevovode međusobno povezanih komponenti koje rade zajedno kako bi postigle inteligentno ponašanje.

U svojoj srži, složeni AI sistemi sastoje se od više modula, od kojih je svaki dizajniran za izvođenje specifičnih zadataka ili funkcija. Ovi moduli mogu uključivati generatore, pretraživače, rangere, klasifikatore i razne druge specijalizovane komponente. Razbijanjem ukupnog sistema na manje, fokusirane jedinice, programeri mogu kreirati fleksibilnije, skalabilnije i održivije AI arhitekture.

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

Međutim, izgradnja složenih AI sistema također predstavlja jedinstvene izazove. Posebno, osiguravanje sveukupne koherentnosti i dosljednosti u ponašanju sistema zahtijeva robusno testiranje, praćenje i mehanizme upravljanja.

Pojava moćnih LLM-ova poput GPT-4 nam omogućava da eksperimentišemo sa složenim AI sistemima lakše nego ikad prije, jer su ovi napredni modeli sposobni da obavljaju višestruke uloge unutar složenog sistema, kao što su klasifikacija, rangiranje i generisanje, pored njihovih sposobnosti razumijevanja prirodnog jezika. Ova svestranost omogućava programerima da brzo kreiraju prototipove i iteriraju na arhitekturama složenih AI sistema, otvarajući nove mogućnosti za razvoj inteligentnih aplikacija.

Obrasci Implementacije za Složene AI Sisteme

Složeni AI sistemi mogu biti implementirani koristeći različite obrasce, od kojih je svaki dizajniran da odgovori na specifične zahtjeve i slučajeve upotrebe. Istražimo četiri

uobičajena obrasca implementacije: Pitanja i Odgovori, Višeagentni/Agentski Rješavači Problema, Konverzacijski AI i CoPiloti.

Pitanja i Odgovori

Sistemi za pitanja i odgovore (Q&A) fokusiraju se na pružanje pretraživanja informacija koje je poboljšano sposobnostima razumijevanja AI modela kako bi funkcionisali kao više od običnog pretraživača. Kombinovanjem moćnih jezičkih modela sa vanjskim izvorima znanja koristeći Generisanje Potpomognuto Pretraživanjem (RAG), sistemi za pitanja i odgovore izbjegavaju halucinacije i pružaju tačne i kontekstualno relevantne odgovore na upite korisnika.

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

- **Razumijevanje i preformulacija upita**: Analiziranje korisničkih upita i njihovo preformulisanje kako bi se bolje podudarali sa osnovnim izvorima znanja.
- **Pretraživanje znanja**: Preuzimanje relevantnih informacija iz strukturiranih ili nestrukturiranih izvora podataka na osnovu preformulisanog upita.
- **Generisanje odgovora**: Generisanje koherentnih i informativnih odgovora integrisanjem preuzetog znanja sa generativnim sposobnostima jezičkog modela.

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

Višeagentni/Agentski Rješavači Problema

Višeagentni, također poznati kao *Agentski*, sistemi sastoje se od više autonomnih agenata koji rade zajedno na rješavanju složenih problema. Svaki agent ima specifičnu ulogu, set vještina i pristup relevantnim alatima ili izvorima informacija. Kroz saradnju i razmjenu informacija, ovi agenti mogu rješavati zadatke koji bi bili teški ili nemogući za jednog agenta da ih riješi sam.

Ključni principi višeagentnih rješavača problema uključuju:

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

Višeagentni sistemi su pogodni za aplikacije koje zahtijevaju distribuirano rješavanje problema, kao što su optimizacija lanca snabdijevanja, upravljanje saobraćajem, ili planiranje odgovora na vanredne situacije

Konverzacijski AI

Konverzacijski AI sistemi omogućavaju interakcije prirodnim jezikom između korisnika i inteligentnih agenata. Ovi sistemi kombinuju razumijevanje prirodnog jezika, upravljanje dijalogom i sposobnosti generisanja jezika kako bi pružili angažirajuća i personalizovana konverzacijska iskustva.

Glavne komponente konverzacijskog AI sistema uključuju:

- **Prepoznavanje namjere**: Identifikovanje namjere korisnika na osnovu njihovog unosa, kao što je postavljanje pitanja, pravljenje zahtjeva ili izražavanje osjećanja.
- **Ekstrakcija entiteta**: Izvlačenje relevantnih entiteta ili parametara iz korisničkog unosa, kao što su datumi, lokacije ili nazivi proizvoda.
- **Upravljanje dijalogom**: Održavanje stanja konverzacije, određivanje odgovarajućeg odgovora na osnovu namjere korisnika i konteksta, te upravljanje interakcijama koje se odvijaju u više koraka.
- **Generisanje odgovora**: Generisanje odgovora nalik ljudskim koristeći jezičke modele, predloške ili metode zasnovane na pretraživanju.

Konverzacijski AI sistemi se obično koriste u četbotovima za korisničku podršku, virtuelnim asistentima i interfejsima kontrolisanim glasom. Kao što je ranije spomenuto, većina pristupa, obrazaca i primjera koda u ovoj knjizi su direktno izvučeni iz mog rada na velikom konverzacijskom AI sistemu koji se zove Olympia

CoPiloti

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

Ključne karakteristike CoPilota uključuju:

- **Personalizacija**: Prilagođavanje individualnim korisničkim preferencijama, radnim tokovima i stilovima komunikacije.
- **Proaktivna pomoć**: Predviđanje korisničkih potreba i nuđenje relevantnih prijedloga ili akcija bez eksplicitnih upita.
- **Kontinuirano učenje**: Poboljšanje performansi tokom vremena učenjem iz povratnih informacija korisnika, interakcija i podataka.

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

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

Uloge u složenim AI sistemima

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

Generator

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

Retriever

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

Ranker

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

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

Classifier

Classifieri se koriste za kategorizaciju ili označavanje podataka na osnovu predefiniranih klasa ili kategorija. Oni uče iz označenih podataka za treniranje i zatim predviđaju klasu novih, neviđenih primjera. Classifieri su fundamentalni za zadatke poput analize sentimenta, detekcije spama ili prepoznavanja slika, gdje je cilj dodijeliti specifičnu kategoriju svakom ulazu.

Alati i Agenti

Pored ovih osnovnih uloga, složeni AI sistemi 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, kao što su generatori ili retrieveri, kako bi izvršili podzadatke ili prikupili dodatne informacije. Primjeri alata uključuju web pretraživače, kalkulatore ili biblioteke za vizualizaciju podataka.
- **Agenti**: Agenti su autonomni entiteti koji mogu percipirati svoje okruženje, donositi odluke i poduzimati akcije kako bi postigli specifične ciljeve. Oni se često oslanjaju na kombinaciju različitih AI tehnika, kao što su planiranje, rasuđivanje i učenje, kako bi efikasno djelovali u dinamičnim ili neizvjesnim uslovima. Agenti se mogu koristiti za modeliranje složenih ponašanja ili za koordinaciju akcija više modula unutar složenog AI sistema.

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

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

Korištenjem snage ovih komponenti i njihovih interakcija, složeni AI sistemi 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 korištene u složenim AI sistemima mogu primijeniti za stvaranje inteligentnih, adaptivnih i korisnički orijentiranih aplikacija.

U narednim poglavljima Dijela 1, dublje ćemo zaroniti u fundamentalne pristupe i tehnike za integraciju AI komponenti u vaš proces razvoja aplikacija. Od inženjeringa upita i generiranja potpomognutog pretraživanjem do samopopravljajućih podataka i inteligentne orkestracije radnih tokova, pokrit ćemo širok raspon obrazaca i najboljih praksi kako bismo vam pomogli da izgradite najsavremenije AI-pogonske aplikacije.

Dio 1: Osnovni pristupi i tehnike

Ovaj dio knjige predstavlja različite načine integracije AI-ja u vaše aplikacije. Poglavlja obuhvataju niz povezanih pristupa i tehnika, od konceptualnijih pojmova kao što su Sužavanje puta i Retrieval Augmented Generation, pa sve do ideja za programiranje vlastitog apstraktnog sloja iznad API-ja za dovršavanje LLM razgovora.

Cilj ovog dijela knjige je da vam pomogne razumjeti vrste ponašanja koje možete implementirati pomoću AI-ja, prije nego što dublje zađemo u specifične implementacijske obrasce koji su fokus Dijela 2.

Pristupi u Dijelu 1 zasnovani su na idejama koje sam koristio u svom kodu, klasičnim obrascima arhitekture poslovnih aplikacija i integracije, plus metaforama koje sam koristio pri objašnjavanju mogućnosti AI-ja drugim ljudima, uključujući i poslovne dionike koji nisu tehnički obrazovani.

Suziti Put

“Suziti put” se odnosi na fokusiranje vještačke inteligencije na zadatak koji je pred njom. Koristim to kao mantru kad god postanem frustriran zbog toga što se AI ponaša “glupo” ili na neočekivan način. Mantra me podsjeća da je neuspjeh vjerovatno moja greška i da vjerovatno trebam još više suziti put.

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

Pristup tako širokom spektru znanja je nesumnjivo moćan i proizvodi emergentno ponašanje kao što su teorija uma i sposobnost rasuđivanja na ljudski način. Međutim, ta zapanjujuća količina informacija također predstavlja izazove kada je u pitanju generiranje preciznih i tačnih odgovora na specifične promptove, posebno ako ti promptovi trebaju pokazivati deterministično ponašanje koje se može integrisati s "normalnim" razvojem softvera i algoritmima.

Nekoliko faktora dovodi do ovih izazova.

Preopterećenje informacijama: Veliki jezički modeli su trenirani na ogromnim količinama podataka koji obuhvataju različite domene, izvore i vremenske periode. Ovo opsežno znanje im omogućava da se bave različitim temama i generišu odgovore zasnovane na širokom razumijevanju svijeta. Međutim, kada se suoči sa specifičnim promptom, model se može mučiti da filtrira irelevantne, kontradiktorne ili zastarjele informacije, što dovodi do odgovora koji nemaju fokus ili tačnost. Zavisno od toga što pokušavate postići, sama količina *kontradiktornih* informacija dostupnih modelu može lako nadvladati njegovu sposobnost da pruži odgovor ili ponašanje koje tražite.

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

Napomena: Kada tek počinjete s "inženjerstvom promptova", mnogo je vjerovatnije 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 jezički modeli trenirani na podacima koji su

kreirani u različitim vremenskim periodima, mogu posjedovati znanje koje je zastarjelo, prevaziđeno ili više nije tačno. Na primjer, informacije o trenutnim događajima, naučnim otkrićima ili tehnološkim naprecima možda su se razvile od vremena kada su prikupljeni podaci za trening modela. Bez sužavanja puta da se prioritetizuju noviji i pouzdaniji izvori, model može generisati odgovore zasnovane na zastarjelim ili netačnim informacijama, što dovodi do netačnosti i nedosljednosti u njegovim izlazima.

Specifičnosti domena: Različite domene i polja imaju svoju specifičnu terminologiju, konvencije i baze znanja. Razmislite o praktično bilo kojem TLA (akronimu od tri slova) i shvatit ćete da većina njih ima više od jednog značenja. Na primjer, MSK se može odnositi na Amazon-ov Managed Streaming for Apache Kafka, Memorial Sloan Kettering Cancer Center, ili ljudski MusculoSKeletal sistem.

Kada prompt zahtijeva stručnost u određenoj domeni, generičko znanje velikog jezičkog modela možda neće biti dovoljno za pružanje tačnih i nijansiranih odgovora. Sužavanje puta fokusiranjem na informacije specifične za domenu, bilo kroz inženjerstvo promptova ili generaciju potpomognutu pronalaženjem, omogućava modelu da generiše odgovore koji su više usklađeni sa zahtjevima i očekivanjima vaše specifične domene.

Latentni prostor: Neshvatljivo prostran

Kada spominjem "latentni prostor" jezičkog modela, govorim o ogromnom, višedimenzionalnom pejzažu znanja i informacija koje je model naučio tokom procesa treninga. To je poput skrivenog carstva unutar neuronskih mreža modela, gdje su pohranjeni svi obrasci, asocijacije i reprezentacije jezika.

Zamislite da istražujete ogromnu, neistraženu teritoriju ispunjenu bezbrojnim međusobno povezanim čvorovima. Svaki čvor predstavlja dio informacije, koncept ili vezu koju je model naučio. Dok se krećete kroz ovaj prostor, primijetit ćete da su neki čvorovi bliže jedni drugima, što ukazuje na jaku povezanost ili sličnost, dok su drugi udaljeniji, što sugeriše slabiju ili dalju vezu.

Izazov sa latentnim prostorom je što je nevjerovatno kompleksan i visokodimenzionalan. Zamislite ga kao naš fizički univerzum, sa njegovim klasterima galaksija i ogromnim, nezamislivim razdaljinama praznog prostora između njih.

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

Stvar je u tome što je model naučio ogromnu količinu informacija iz svojih podataka za treniranje, i nisu sve relevantne ili tačne za određeni zadatak. Zato sužavanje puta postaje tako važno. Pružanjem jasnih instrukcija, primjera i konteksta u svojim promptovima, vi u suštini usmjeravate model da se fokusira na specifične regije unutar latentnog prostora koje su najrelevantnije za vaš željeni izlaz.

Drugačiji način da to zamislite 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 bezbrojnim objektima i detaljima. Vaš prompt je reflektor, osvjetljavajući specifična područja i usmjeravajući pažnju modela na najvažnije informacije. Bez tog vođenja, model može besciljno lutati kroz latentni prostor, skupljajući irelevantne ili kontradiktorne informacije usput.

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

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

U svojoj srži, jezički modeli su probabilistički modeli teksta, slično kao što je Gausova kriva statistički model podataka. Oni se treniraju kroz proces koji se zove autoregresivno modeliranje, gdje model uči da predvidi vjerovatnoću sljedeće riječi u sekvenci na osnovu riječi koje joj prethode. Tokom treniranja, model počinje sa nasumičnim težinama i postepeno ih prilagođava kako bi dodijelio veće vjerovatnoće tekstu koji liči na stvarne uzorke na kojima je treniran.

Međutim, razmišljanje o jezičkim 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 omogućavaju manipulaciju slučajnim varijablama i mogu predstavljati složene statističke odnose.

Probabilistički programi mogu biti predstavljeni grafičkim modelima, koji pružaju vizuelni način za razumijevanje zavisnosti i odnosa između varijabli u modelu. Ova perspektiva može ponuditi vrijedne uvide u rad složenih modela za generisanje teksta poput GPT-4 i Claude.

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

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

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

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

Umjesto toga, snaga modela leži u njegovoj sposobnosti da pronađe i sintetizira informacije koje su slične onima koje mu zadajemo kroz upit. Razumijevanjem probabilističke prirode ovih modela i načina na koji se upiti mogu koristiti za uvjetovanje njihovih izlaza, možemo efikasnije iskoristiti njihove mogućnosti za generiranje vrijednih uvida i sadržaja.

Razmotrite upite u nastavku. U prvom, sama riječ "Merkur" može se odnositi na planetu, hemijski element ili rimskog boga, ali najvjerovatnije je da se radi o planeti. Zaista, GPT-4 daje opširan odgovor koji počinje sa *Merkur je najmanja i Suncu najbliža planeta Sunčevog sistema.*... Drugi upit se konkretno odnosi na hemijski 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 kako AI reaguje. Kao što ćete kasnije naučiti u knjizi, napredni trikovi u kreiranju promptova kao što su n-shot promptovanje, strukturirani unos/izlaz i Lanac Razmišljanja su samo pametni načini uslovljavanja izlaza modela.

Dakle, u suštini, umjetnost inženjeringa promptova se svodi na razumijevanje kako navigirati kroz ogromni probabilistički pejzaž znanja jezičkog modela kako bismo suzili put do specifičnih informacija ili ponašanja koje tražimo.

Za čitaoce sa dobrim razumijevanjem napredne matematike, temeljenje vašeg razumijevanja ovih modela na principima teorije vjerovatnoće i linearne algebre definitivno može pomoći! Za ostale koji žele razviti efektivne strategije za dobijanje željenih izlaza, držat ćemo se intuitivnijih pristupa.

Kako Se Put "Sužava"

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

Evo najznačajnijih tehnika, u preporučenom redoslijedu, odnosno, trebali biste prvo pokušati s Inženjeringom promptova, zatim RAG, i konačno, ako morate, fino podešavanje.

Inženjering promptova Najosnovniji pristup je kreiranje promptova koji uključuju specifične instrukcije, ograničenja ili primjere za usmjeravanje generisanja odgovora modela. Ovo poglavlje pokriva osnove Inženjeringa promptova u sljedećem odjeljku, a mnoge specifične obrasce inženjeringa promptova obrađujemo u Dijelu 2 knjige. Ti obrasci uključuju Destilaciju promptova, tehniku koja se fokusira na rafiniranje i optimizaciju promptova za izvlačenje onoga što AI smatra najrelevantnijim i najsažetijim informacijama.

Proširivanje konteksta. Dinamičko preuzimanje relevantnih informacija iz vanjskih baza znanja ili dokumenata kako bi se modelu pružio fokusirani kontekst u trenutku kada se promptuje. Popularne tehnike proširivanja konteksta uključuju Generisanje potpomognuto preuzimanjem (RAG) Takozvani "online modeli" poput onih koje pruža Perplexity mogu proširiti svoj kontekst rezultatima pretraživanja interneta u stvarnom vremenu.

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

Fino podešavanje ili Adaptacija domene Treniranje modela na skupovima podataka specifičnim za domenu kako bi se specijaliziralo njegovo znanje i sposobnosti generisanja za određeni zadatak ili oblast.

Smanjivanje Temperature

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

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

S druge strane, kada je temperatura postavljena na 0, jezički model uvijek bira token s najvišom vjerovatnoćom, 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 *prati instrukcije*, obrati pažnju na funkcije koje su mu pružene, ili jednostavno trebate tačnije i relevantnije odgovore od onih koje dobijate.

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

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

Hiperparametri: Dugmad i Prekidači Zaključivanja

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

Zamislite to kao podešavanje postavki na složenom uređaju. Baš kao što biste mogli okrenuti dugme da kontrolišete temperaturu ili prebaciti prekidač da promijenite način rada, hiperparametri vam omogućavaju fino podešavanje načina na koji jezički model obrađuje i generiše tekst.

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

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

- **Top-p (nucleus) uzorkovanje**: Ovaj parametar kontroliše odabir najmanjeg skupa tokena čija kumulativna vjerovatnoća prelazi određeni prag (p). Omogućava raznovrsnije izlaze dok istovremeno održava koherentnost.

- **Top-k uzorkovanje**: Ova tehnika odabire k najvjerovatnijih sljedećih tokena i preraspodjeljuje vjerovatnoću među njima. Može pomoći u sprječavanju modela da generiše tokene male vjerovatnoće ili irelevantne tokene.

- **Kazne za učestalost** i **prisustvo**: Ovi parametri kažnjavaju model za prečesto ponavljanje istih riječi ili fraza (kazna za učestalost) ili za generisanje riječi

koje nisu prisutne u ulaznom promptu (kazna za prisustvo). Podešavanjem ovih vrijednosti možete potaknuti model da proizvodi raznovrsnije i relevantnije izlaze.

- **Maksimalna dužina**: Ovaj hiperparametar postavlja gornju granicu na broj tokena (riječi ili podriječi) koje model može generisati u jednom odgovoru. Pomaže u kontroli opširnosti i konciznosti generisanog teksta.

Dok eksperimentišete s različitim postavkama hiperparametara, primijetit ćete da čak i male prilagodbe mogu imati značajan uticaj na izlaz modela. To je poput finog podešavanja recepta – prstohvat više soli ili malo duže vrijeme kuhanja mogu napraviti veliku razliku u konačnom jelu.

Ključ je razumjeti kako svaki hiperparametar utiče na ponašanje modela i pronaći pravu ravnotežu za vaš specifični zadatak. Ne bojte se eksperimentisati s različitim postavkama i vidjeti kako one utiču na generisani tekst. Vremenom ćete razviti intuiciju za to koje hiperparametre treba prilagoditi i kako postići željene rezultate.

Kombinovanjem korištenja ovih parametara s inženjeringom promptova, generisanjem potpomognutim pretraživanjem i finim podešavanjem, možete efikasno suziti put i voditi jezični model da generiše preciznije, relevantnije i vrednije odgovore za svoj specifični slučaj upotrebe.

Osnovni naspram modela podešenih za instrukcije

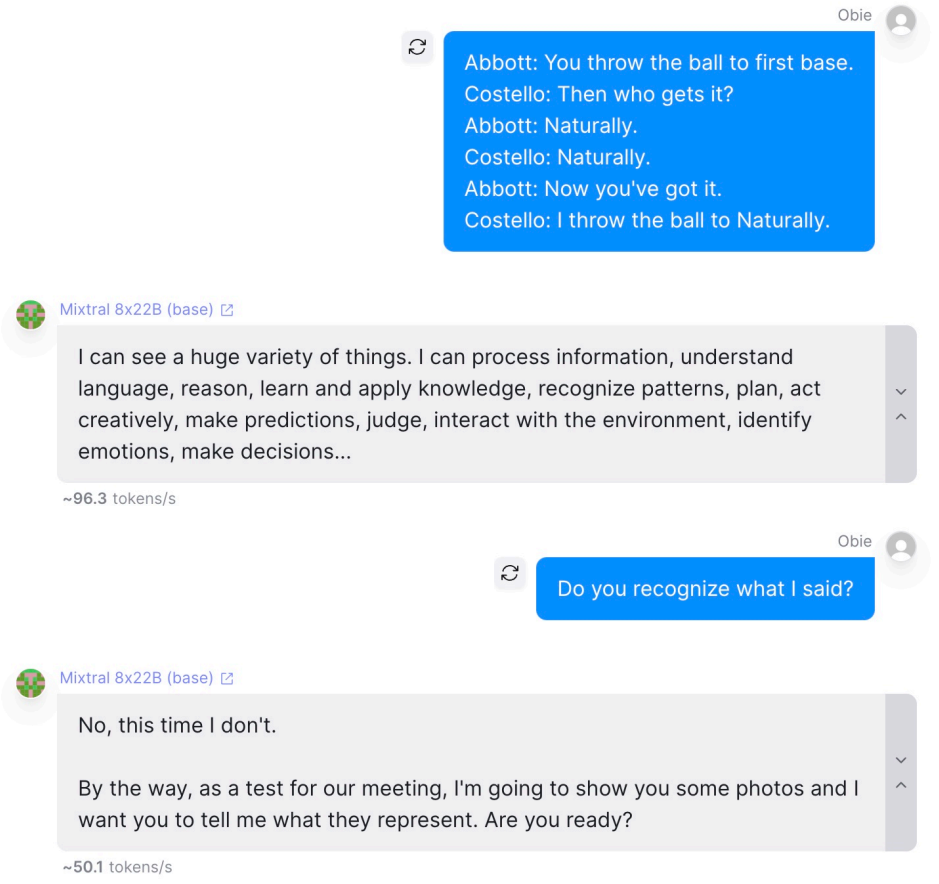
Osnovni modeli su nerefinisane, netrenirane verzije LLM-ova. Zamislite ih kao prazno platno, još uvijek nepod uticajem specifičnog treninga za razumijevanje ili praćenje instrukcija. Izgrađeni su na osnovu ogromnih podataka na kojima su inicijalno trenirani, sposobni za generisanje širokog spektra izlaza. Međutim, bez dodatnih slojeva finog podešavanja baziranog na instrukcijama, njihovi odgovori mogu biti nepredvidljivi i

zahtijevaju više nijansiranih, pažljivo izrađenih promptova da ih vode prema željenom izlazu. Rad s osnovnim modelima je poput izvlačenja komunikacije iz idiot-savanta koji ima ogromnu količinu znanja ali nema nikakvu intuiciju o tome što ga pitate osim ako niste izuzetno precizni u svojim instrukcijama. Često se čine poput papagaja, u smislu da u mjeri u kojoj ih natjerate da kažu nešto razumljivo, najčešće samo ponavljaju nešto što su čuli da ste rekli.

S druge strane, modeli podešeni za instrukcije prošli su kroz runde treninga posebno dizajnirane za razumijevanje i praćenje instrukcija. GPT-4, Claude 3 i mnogi drugi od najpopularnijih LLM modela su svi intenzivno podešeni za instrukcije. Ovaj trening uključuje hranjenje modela primjerima instrukcija zajedno sa željenim ishodima, efektivno učeći model kako da interpretira i izvršava širok spektar naredbi. Kao rezultat, instrukcijski modeli mogu lakše razumjeti namjeru iza prompta i generisati odgovore koji se blisko usklađuju s očekivanjima korisnika. Ovo ih čini pristupačnijim i lakšim za rad, posebno za one koji možda nemaju vremena ili stručnosti za ekstenzivni inženjering promptova.

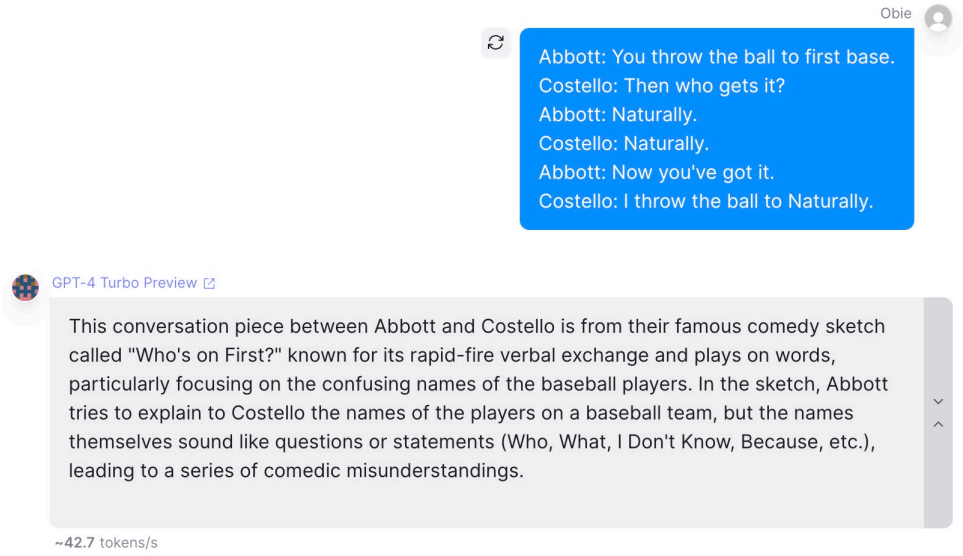
Osnovni modeli: Nefiltrirano platno

Osnovni modeli, kao što su Llama 2-70B ili Yi-34B, nude više nefiltriranog pristupa mogućnostima modela od onoga na što ste možda navikli ako ste eksperimentisali s popularnim LLM-ovima poput GPT-4. Ovi modeli nisu unaprijed podešeni da prate specifične instrukcije, pružajući vam prazno platno za direktnu manipulaciju izlaza modela kroz pažljivi inženjering promptova. Ovaj pristup zahtijeva duboko razumijevanje kako kreirati promptove koji vode AI u željenom smjeru bez eksplicitnog instruiranja. To je slično kao da imate direktan pristup "sirovim" slojevima osnovnog AI-ja, bez ikakvih posredničkih slojeva koji interpretiraju ili vode odgovore modela (otuda i ime).



Slika 3. Testiranje osnovnog modela koristeći dio Abbott i Costello klasičnog skeča 'Ko je na prvoj bazi'

Izazov s osnovnim modelima leži u njihovoj tendenciji da upadnu u repetitivne obrasce ili proizvode nasumični output. Međutim, uz pažljivo osmišljavanje promptova i podešavanje parametara kao što su kazne ponavljanja, osnovni modeli se mogu navesti da generišu jedinstveni i kreativni sadržaj. Ovaj proces nije bez kompromisa; dok osnovni modeli nude nenadmašnu fleksibilnost za inovacije, oni zahtijevaju viši nivo stručnosti.



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

Modeli Podešeni za Instrukcije: Vođeno Iskustvo

Modeli podešeni za instrukcije su dizajnirani da razumiju i slijede specifične instrukcije, čineći ih pristupačnijim i dostupnijim za širi spektar aplikacija. Oni razumiju mehaniku *razgovora* i znaju da trebaju prestati generisati kada je *kraj njihovog reda za razgovor*. Za mnoge programere, posebno one koji rade na jednostavnim aplikacijama, modeli podešeni za instrukcije nude praktično i efikasno rješenje.

Proces podešavanja instrukcija uključuje treniranje modela na velikom korpusu instrukcijskih promptova i odgovora koje su generisali ljudi. Jedan značajan primjer je databricks-dolly-15k dataset otvorenog koda, koji sadrži preko 15.000 parova promptova/odgovora koje su kreirali Databricks zaposlenici koje možete sami pregledati. Dataset pokriva osam različitih kategorija instrukcija, uključujući kreativno pisanje, odgovaranje na zatvorena i otvorena pitanja, sumiranje, ekstrakciju informacija, klasifikaciju, i generisanje ideja.

Tokom procesa generisanja podataka, saradnicima su date smjernice o tome kako kreirati promptove i odgovore za svaku kategoriju. Na primjer, za zadatke kreativnog pisanja, dobili su upute da osiguraju specifična ograničenja, instrukcije ili zahtjeve koji će voditi output modela. Za odgovaranje na zatvorena pitanja, zatraženo im je da napišu pitanja koja zahtijevaju faktički tačne odgovore bazirane na datom Wikipedia odlomku.

Rezultirajući dataset služi kao vrijedan resurs za fino podešavanje velikih jezičkih modela kako bi pokazivali interaktivne sposobnosti i sposobnosti praćenja instrukcija sistema poput ChatGPT-a. Treniranjem na raznovrsnom spektru instrukcija i odgovora koje su generisali ljudi, model uči razumjeti i slijediti specifične direktive, čineći ga sposobnijim za rukovanje širokim spektrom zadataka.

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

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

Jednostavnost korištenja koju pružaju modeli podešeni na instrukcije dolazi uz cijenu određene fleksibilnosti. Ovi modeli su često snažno cenzurirani, što znači da možda neće uvijek pružiti nivo kreativne slobode potreban za određene zadatke. Na njihove izlaze snažno utiču pristrasnosti i ograničenja svojstvena podacima korištenim za njihovo fino podešavanje.

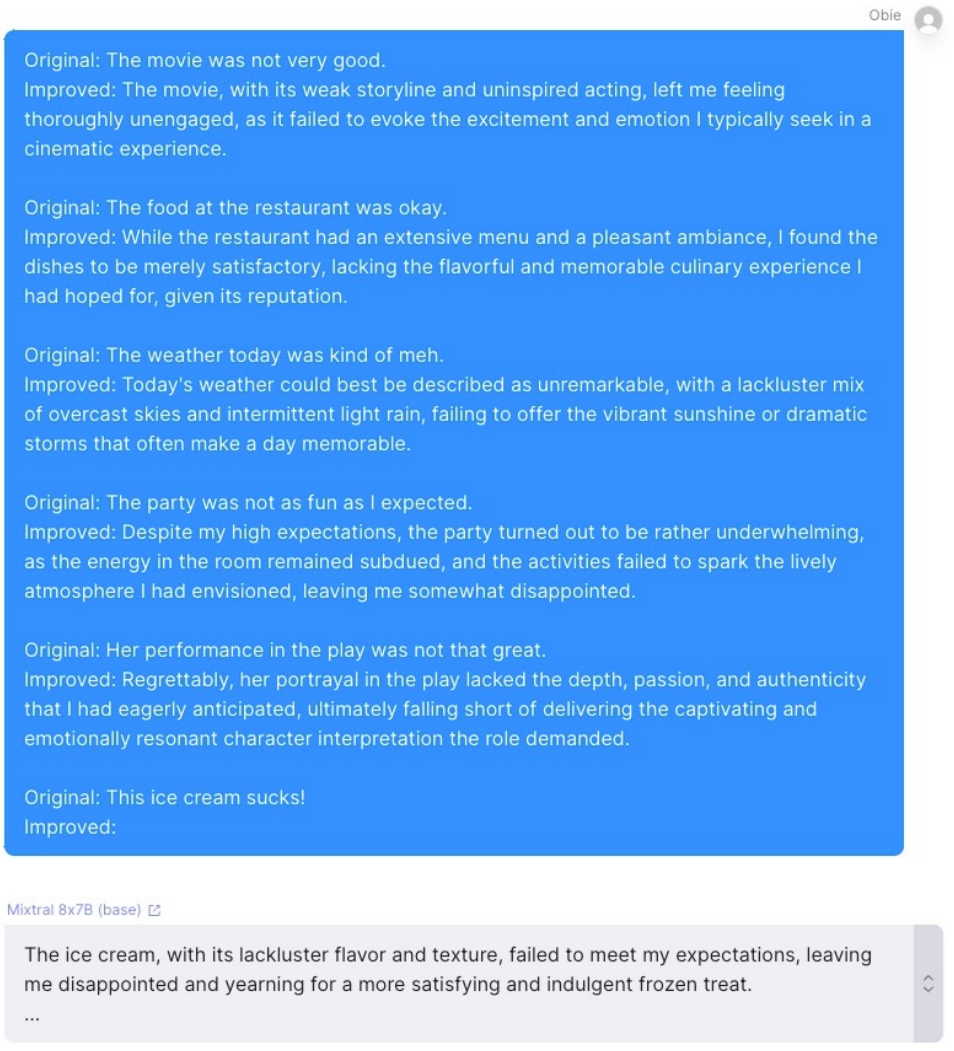
Uprkos ovim ograničenjima, modeli podešeni na instrukcije postali su sve popularniji zbog svoje pristupačnosti korisnicima i sposobnosti da se nose sa širokim spektrom zadataka uz minimalno inženjerstvo promptova. Kako više visokokvalitetnih skupova

podataka s uputstvima postaje dostupno, možemo očekivati daljnja poboljšanja u performansama i svestranosti ovih modela.

Odabir pravog tipa modela za vaš projekat

Odluka između osnovnih (sirovih) i modela podešenih na instrukcije u konačnici zavisi od specifičnih zahtjeva vašeg projekta. Za zadatke koji zahtijevaju visok stepen kreativnosti i originalnosti, osnovni modeli nude moćan alat za inovacije. Ovi modeli omogućavaju programerima da istraže puni potencijal LLM-ova, pomjerajući granice onoga što se može postići kroz aplikacije vođene vještačkom inteligencijom, ali zahtijevaju direktniji pristup i spremnost na eksperimentisanje. Temperatura i druge postavke imaju mnogo veći efekat u osnovnim modelima nego u njihovim instrukcijskim pandanima.

Šta god uključite u svoj prompt je ono što će osnovni modeli pokušati ponoviti. Tako da ako je vaš prompt, na primjer, transkript chata, sirovi model će pokušati nastaviti chat. Zavisno od ograničenja maksimalnog broja tokena, neće samo generisati sljedeću poruku u chatu, može voditi cijelu konverzaciju sam sa sobom!



Slika 5. Mixtral 8x7B (base) Primjer prepisivanja rečenica s few-shot završetkom

Dok sam pripremao gornji primjer Prepisivanja rečenica Reddit korisnika phree_radical, uspio sam dobiti upotrebljive rezultate tek nakon mnogo eksperimentisanja 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 težak zbog snažnog efekta parametra `max_tokens`. Ako ga postavite prekratko, izlaz će biti odsječen. Ako ga postavite duže nego što je modelima potrebno za željeni izlaz, nastavit će halucinirati dodatne primjere.

Suština je da, osim ako vam stvarno ne treba potpuna kontrola i odsustvo cenzure, modeli podešeni na instrukcije mogu značajno pojednostaviti vaš proces razvoja. Da bismo to dodatno naglasili, evo odgovora Mixtral 8x7B na isti prompt, ali ovog puta u njegovoj verziji podešenoj na instrukcije:

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

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

Prompt Inženjering

Kada počnete primjenjivati AI u svojim projektima, brzo ćete otkriti da je jedna od najvažnijih vještina koju trebate savladati umjetnost prompt inženjeringa. Ali šta je zapravo prompt inženjering i zašto je toliko važan?

U svojoj suštini, prompt inženjering je proces dizajniranja i oblikovanja ulaznih promptova koje dajete jezičkom modelu kako biste usmjerili njegov izlaz. Radi se o razumijevanju kako efikasno komunicirati sa AI-jem, koristeći kombinaciju instrukcija, primjera i konteksta da biste usmjerili model prema generisanju željenog odgovora.

Zamislite to kao razgovor sa 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 tačno razumije šta tražite. Tu nastupa prompt inženjering, i iako na prvi pogled djeluje jednostavno, vjerujte mi da je potrebno mnogo prakse da biste ga savladali.

Gradivni Elementi Efektivnih Promptova

Da biste počeli kreirati efektivne promptove, prvo morate razumjeti ključne komponente koje čine dobro oblikovan unos. Evo nekih od osnovnih gradivnih elemenata:

1. **Instrukcije**: Jasne i koncizne upute koje govore modelu šta želite da uradi. To može biti bilo šta, od "Sumirati sljedeći članak" do "Generisati pjesmu o zalasku sunca" do "pretvoriti ovaj zahtjev za promjenu projekta u JSON objekat".
2. **Kontekst**: Relevantne informacije koje pomažu modelu da razumije pozadinu i opseg zadatka. Ovo može uključivati detalje o ciljanoj publici, željenom tonu i stilu, ili bilo koje specifične zahtjeve ili ograničenja za izlaz, kao što je JSON Schema kojoj se treba pridržavati.
3. **Primjeri**: Konkretni primjeri koji demonstriraju tip izlaza koji tražite. Pružanjem nekoliko dobro odabranih primjera možete pomoći modelu da nauči obrasce i karakteristike željenog odgovora.
4. **Formatiranje Unosa**: Prelomi redova i markdown formatiranje daju strukturu našem promptu. Odvajanje prompta u paragrafe nam omogućava da grupišemo povezane instrukcije tako da ih i ljudi i AI lakše razumiju. Tačke i numerisane liste nam omogućavaju da definišemo liste i redoslijed stavki. Oznake za podebljano i kurziv nam omogućavaju da naglasimo važne dijelove.
5. **Formatiranje Izlaza**: Specifične instrukcije o tome kako izlaz treba biti strukturiran i formatiran. Ovo može uključivati direktive o željenoj dužini, korištenju naslova ili tačaka, markdown formatiranju, ili bilo kojim drugim specifičnim šablonima ili konvencijama izlaza kojih se treba pridržavati.

Kombinovanjem ovih gradivnih elemenata na različite načine možete kreirati promptove koji su prilagođeni vašim specifičnim potrebama i usmjeravaju model prema generisanju kvalitetnih, relevantnih odgovora.

Umjetnost i Nauka Dizajna Promptova

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

Jedan od ključnih aspekata prompt inženjeringa 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 preskriptivni da ograničite sposobnost modela da koristi vlastitu kreativnost i fleksibilnost u rješavanju graničnih slučajeva.

Još jedan važan faktor je upotreba primjera. Dobro odabrani primjeri mogu biti nevjerovatno moćni u pomaganju modelu da razumije tip izlaza koji tražite. Međutim, važno je primjere koristiti 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 Prompt Inženjeringa

Kako budete dublje ulazili u svijet prompt inženjeringa, otkrit ćete niz tehnika i najboljih praksi koje vam mogu pomoći da kreirate efektivnije promptove. Evo nekoliko ključnih područja za istraživanje:

1. **Zero-shot nasuprot few-shot učenju**: Razumijevanje kada koristiti *zero-shot* učenje (bez pružanja primjera) nasuprot *one-shot* ili *few-shot* učenju (pružanje

malog broja primjera) može vam pomoći da kreirate promptove koji su efikasniji i efektivniji.

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

Učenje kada koristiti zero-shot, one-shot ili few-shot učenje je posebno važan dio savladavanja inženjeringa promptova. Svaki pristup ima svoje prednosti i mane, a razumijevanje kada koristiti koji može vam pomoći da kreirate efikasnije i učinkovitije promptove.

Zero-Shot učenje: Kada primjeri nisu potrebni

Zero-shot učenje odnosi se na sposobnost jezičnog modela da izvrši zadatak bez ikakvih primjera ili eksplicitnog treninga. Drugim riječima, modelu dajete prompt koji opisuje zadatak, a model generira odgovor isključivo na osnovu svog postojećeg znanja i razumijevanja jezika.

Zero-shot učenje je posebno korisno kada:

1. Je zadatak relativno jednostavan i jasan, a model je vjerojatno naišao na slične zadatke tijekom svog pred-treninga.

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

Međutim, zero-shot učenje može biti nepredvidljivo i ne mora uvijek proizvesti željene rezultate. Na odgovor modela mogu utjecati pristrasnosti ili nedosljednosti u podacima za pred-trening, a model se može mučiti sa složenijim ili nijansiranijim zadacima.

> Vidio sam zero-shot promptove koji rade dobro za 80% mojih testnih slučajeva i proizvode potpuno pogrešne ili nerazumljive rezultate za ostalih 20%. Vrlo je važno implementirati temeljit režim testiranja, posebno ako se značajno oslanjate na zero-shot promptovanje.

One-Shot učenje: Kada jedan primjer može napraviti razliku

One-shot učenje 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.

One-shot učenje može biti efikasno kada:

1. Je zadatak relativno nov ili specifičan, a model možda nije naišao na mnogo sličnih primjera tijekom pred-treninga.
2. Želite pružiti jasan i koncizan prikaz željenog formata ili stila izlaza.
3. Zadatak zahtijeva specifičnu strukturu ili konvenciju koja možda nije očigledna samo iz opisa zadatka.

Opisi koji su vama očigledni možda nisu nužno očigledni AI-u. One-shot primjeri mogu pomoći u razjašnjavanju.

One-shot učenje može pomoći modelu da jasnije razumije očekivanja i generira odgovor koji je više usklađen s datim primjerom. Međutim, važno je pažljivo odabrati primjer i osigurati da je reprezentativan za željeni izlaz. Prilikom odabira primjera, zapitajte se o potencijalnim graničnim slučajevima i rasponu ulaza koje će prompt obrađivati.

Slika 6. One-shot primjer željenog JSON-a

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

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

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

This is an example of well-formed output:

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

Učenje sa malo primjera (Few-Shot Learning): Kada više primjera može poboljšati performanse

Učenje sa malo primjera podrazumijeva davanje modelu malog broja primjera (obično između 2 i 10) zajedno sa opisom zadatka. Ovi primjeri služe da modelu pruže više konteksta i varijacija, pomažući mu da generira raznovrsnije i preciznije odgovore.

Učenje sa malo primjera je posebno korisno kada:

1. Je zadatak složen ili nijansiran, i jedan primjer možda nije dovoljan da obuhvati sve relevantne aspekte.
2. Želite da modelu pružite niz primjera koji pokazuju različite varijacije ili granične slučajeve.
3. Zadatak zahtijeva da model generira odgovore koji su u skladu sa određenom domenom ili stilom.

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

Primjer: Upiti mogu biti mnogo složeniji nego što zamišljate

Današnji veliki jezički modeli su mnogo moćniji i sposobniji za rasuđivanje nego što možete zamisliti. Zato nemojte se ograničavati na razmišljanje o upitima kao o jednostavnoj specifikaciji parova ulaza i izlaza. Možete eksperimentisati sa davanjem dugih i složenih instrukcija na način koji podsjeća na interakciju s ljudima.

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

Započeo sam svoj upit govoreći AI-u da ima direktan pristup Google API krajnjim tačkama putem HTTP-a, i da je njegova uloga da koristi Google aplikacije i servise u ime korisnika. Zatim sam pružio smjernice, pravila vezana za parametar `fields`, pošto se činilo da s tim ima najviše problema, i neke specifične API savjete (few-shot promptovanje na djelu).

Evo cijelog upita, koji govori AI-u kako da koristi 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 da li ovaj prompt funkcioniše. Jednostavan odgovor je da. Umjetna inteligencija nije uvijek znala kako savršeno pozvati API iz prvog pokušaja. Međutim, ako bi napravila grešku, jednostavno bih joj vratio rezultirajuće poruke o grešci kao rezultat poziva. Sa znanjem o svojoj grešci, umjetna inteligencija je mogla razmišljati o svojoj grešci i pokušati ponovo. U većini slučajeva, uspjela bi iz nekoliko pokušaja.

Imajte na umu da su velike JSON strukture koje Google API vraća kao podatke prilikom korištenja ovog prompta izuzetno neefikasne, tako da *ne* preporučujem da koristite ovaj pristup u produkciji. Međutim, mislim da je činjenica da je ovaj pristup uopće funkcionisao dokaz koliko moćan inženjering promptova može biti.

Eksperimentisanje i Iteracija

U konačnici, način na koji ćete dizajnirati svoj prompt zavisi od specifičnog zadatka, složenosti željenog rezultata i mogućnosti jezičkog modela s kojim radite.

Kao inženjer promptova, važno je eksperimentisati s različitim pristupima i iterirati na osnovu 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 da li se performanse poboljšavaju.

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

Vremenom ćete razviti intuiciju za to koji pristup će najbolje funkcionisati za određeni zadatak, i moći ćete kreirati promptove koji su efikasniji i efektivniji. Ključ je ostati radoznao, eksperimentalan i iterativan u svom pristupu inženjeringu promptova.

Kroz ovu knjigu, dublje ćemo zaroniti u ove tehnike i istražiti kako se mogu primijeniti u stvarnim scenarijima. Ovladavanjem umjetnosti i nauke inženjeringa promptova, bićete dobro opremljeni da otključate puni potencijal razvoja aplikacija vođenih umjetnom inteligencijom.

Umjetnost Neodređenosti

Kada je riječ o kreiranju efektivnih promptova za velike jezičke modele (LLM-ove), uobičajena pretpostavka je da više specifičnosti i detaljnih instrukcija vodi do boljih rezultata. Međutim, praktično iskustvo je pokazalo da to nije uvijek slučaj. Zapravo, namjerna neodređenost u vašim promptovima često može dovesti do superiornijih rezultata, koristeći izuzetnu sposobnost LLM-a da generalizuje i izvodi zaključke.

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

kada su u pitanju promptovi. Umjesto preciznih lista ili previše detaljnih instrukcija, Ken je otkrio da dopuštanje LLM-u da se osloni na svoje osnovno znanje često proizvodi bolje rezultate.

Ova spoznaja preokreće tradicionalni način razmišljanja o eksplicitnom kodiranju, gdje sve treba biti detaljno objašnjeno. Sa LLM-ovima, važno je prepoznati da oni posjeduju ogromnu količinu znanja i mogu praviti inteligentne veze i zaključke. Biti neodređeniji u svojim promptovima 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 procesnom toku za klasifikaciju teksta koji se odnosi na jednu od 50 američkih država ili Federalnu vladu, njihov početni pristup je uključivao pružanje *kompletne* detaljne liste država i njihovih odgovarajućih ID-ova kao niz formatiran u JSON-u.

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

Pristup je toliko podbacio da su morali dublje zaći u prompt kako bi otkrili način da ga poboljšaju. Pritom su primijetili da, iako bi VJM često pogrešno naveo id, dosljedno je vraćao puno ime ispravne države u polju `name`, *iako to nisu eksplicitno tražili.*

Uklanjanjem lokalnih id-ova i pojednostavljenjem prompta na nešto poput "Očigledno znaš 50 država, GPT, zato mi samo daj puno ime države na koju se ovo odnosi, ili Federal ako se odnosi na američku vladu," postigli su bolje rezultate. Ovo iskustvo naglašava snagu korištenja sposobnosti generalizacije VJM-a i omogućavanja da donosi zaključke na osnovu postojećeg znanja.

> Kenovo obrazloženje za ovaj poseban pristup klasifikaciji, nasuprot tradicionalnijoj programerskoj tehnici, osvjetljava način razmišljanja onih među nama koji su prihvatili potencijal VJM tehnologije: "Ovo nije težak zadatak – vjerovatno smo mogli koristiti stringove/regex, ali ima dovoljno čudnih graničnih slučajeva da bi to duže trajalo."

Sposobnost VJM-ova da poboljšaju kvalitet i generalizaciju kada dobiju neodređenije promptove je izuzetna karakteristika razmišljanja višeg reda i delegiranja. To pokazuje da VJM-ovi mogu upravljati dvosmislenošću i donositi inteligentne odluke na osnovu datog konteksta.

Međutim, važno je napomenuti da biti neodređen ne znači biti nejasan ili dvosmislen. Ključ je pružiti dovoljno konteksta i smjernica da se VJM usmjeri u pravom smjeru, istovremeno mu omogućavajući fleksibilnost da koristi svoje znanje i sposobnosti generalizacije.

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

1. Fokusirajte 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 zaključke i veze na osnovu datog konteksta.
5. Iterativno poboljšavajte svoje promptove na osnovu odgovora VJM-a, pronalazeći pravu ravnotežu između specifičnosti i neodređenosti.

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

na različite nivoe specifičnosti u vašim promptovima i prilagodite se u skladu s tim. S praksom i iskustvom, razvit ćete istančan osjećaj kada biti neodređeniji, a kada pružiti dodatne smjernice, omogućavajući vam da efikasno iskoristite snagu VJM-ova u vašim aplikacijama.

Zašto antropomorfizam dominira u inženjerstvu promptova

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

Antropomorfizacija VJM-ova pruža okvir koji je odmah intuitivan ljudima koji su potpuno neupućeni u tehničke složenosti sistema. Kao što ćete iskusiti ako pokušate koristiti model koji nije instrukcijski podešen za bilo šta korisno, konstruisanje okvira u kojem očekivani nastavak pruža vrijednost je izazovan zadatak. Zahtijeva prilično duboko razumijevanje unutrašnjeg funkcionisanja sistema, nešto što posjeduje relativno mali broj stručnjaka.

Tretiranjem interakcije s jezičkim modelom kao razgovora između dvije osobe, možemo se osloniti na naše urođeno razumijevanje ljudske komunikacije da prenesemo naše potrebe i očekivanja. Baš kao što je rani Macintosh UI dizajn dao prednost trenutnoj intuitivnosti nad sofisticiranošću, antropomorfni okvir AI-ja nam omogućava da se uključimo na način koji se čini prirodnim i poznatim.

Kada komuniciramo s drugom osobom, naš instinkt je da im se direktno obraćamo koristeći “ti” i dajemo jasne smjernice o tome kako očekujemo da se ponašaju. Ovo se besprijekorno prevodi u proces inženjerstva promptova, gdje usmjeravamo ponašanje AI-ja specificiranjem sistemskih promptova i uključivanjem u dijalog.

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

kognitivno opterećenje i omogućava nam da se fokusiramo na zadatak umjesto da se borimo s tehničkim složenostima sistema.

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

Kako se područje inženjeringa uputa nastavlja razvijati, možemo očekivati daljnja usavršavanja i inovacije u načinu na koji komuniciramo s velikim jezičkim modelima. Međutim, antropomorfizam kao sredstvo za pružanje intuitivnog i pristupačnog iskustva za programere i korisnike će vjerovatno ostati temeljni princip u dizajnu ovih sistema.

Odvajanje instrukcija od podataka: Ključni princip

Neophodno je razumjeti temeljni princip koji podupire sigurnost i pouzdanost ovih sistema: odvajanje instrukcija od podataka.

U tradicionalnoj računarskoj nauci, jasno razlikovanje između pasivnih podataka i aktivnih instrukcija predstavlja osnovni sigurnosni princip. Ovo odvajanje pomaže u sprječavanju neželjenog ili zlonamjernog izvršavanja koda koji bi mogao ugroziti integritet i stabilnost sistema. Međutim, današnji veliki jezički modeli, koji su prvenstveno razvijeni kao modeli koji slijede upute poput chatbotova, često nemaju ovo formalno i principijelno odvajanje.

Što se tiče velikih jezičkih modela, instrukcije se mogu pojaviti bilo gdje u ulaznim podacima, bilo da se radi o sistemskoj uputi ili korisničkoj uputi. Ovaj nedostatak odvajanja može dovesti do potencijalnih ranjivosti i neželjenog ponašanja, slično problemima s kojima se suočavaju baze podataka sa SQL injekcijama ili operativni sistemi bez odgovarajuće zaštite memorije.

Dok radite s velikim jezičkim modelima, ključno je biti svjestan ovog ograničenja i poduzeti korake za ublažavanje rizika. Jedan od pristupa je pažljivo oblikovanje vaših uputa i ulaznih podataka kako bi se jasno razlikovalo između instrukcija i podataka. Tipične metode za pružanje eksplicitnih smjernica o tome šta predstavlja instrukciju, a šta bi se trebalo tretirati kao pasivni podatak, uključuju označavanje markup stilom. Vaša uputa može pomoći velikom jezičkom 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 eskejpovanjem potencijalnih instrukcija ili dijelova koda koji mogu biti ugrađeni u podatke, možete smanjiti šanse za neželjenim izvršavanjem. Obrasci poput Ulančavanja promptova su korisni u ovu svrhu.

Štaviše, dok dizajnirate arhitekturu vaše aplikacije, razmotrite ugradnju mehanizama za provođenje razdvajanja instrukcija i podataka na višem nivou. To može uključivati korištenje zasebnih krajnjih tačaka ili API-ja za rukovanje instrukcijama i podacima, implementaciju stroge validacije i parsiranja ulaznih podataka, te primjenu

principa najmanje privilegije kako bi se ograničio opseg onoga čemu LLM može pristupiti i izvršiti.

Princip najmanje privilegije

Prihvatanje principa najmanje privilegije je poput organizovanja vrlo ekskluzivne zabave gdje gosti dobiju pristup samo onim prostorijama koje su im apsolutno neophodne. Zamislite da ste domaćin ove zabave u prostranoj vili. Ne treba svako da luta po vinskom podrumu ili glavnoj spavaćoj sobi, zar ne? Primjenom ovog principa, vi u suštini 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 igralište. Ako se neko nepozvani ipak ušunja na vašu zabavu, naći će se ograničen na predvorje, takoreći, drastično ograničavajući nestašluk koji može napraviti. Dakle, kada osiguravate svoje LLM aplikacije, zapamtite: dijelite ključeve samo za sobe koje su neophodne, a ostatak vile držite sigurnim. To nije samo stvar lijepog ponašanja; to je dobra sigurnost.

Iako trenutno stanje LLM-ova možda nema formalnu separaciju instrukcija i podataka, za vas kao developera je ključno da budete svjesni ovog ograničenja i da preduzmete proaktivne mjere za ublažavanje rizika. Primjenom najboljih praksi iz računarskih nauka i njihovim prilagođavanjem jedinstvenim karakteristikama LLM-ova, možete izgraditi sigurnije i pouzdanije aplikacije koje koriste snagu ovih modela dok održavaju integritet vašeg sistema.

Destilacija promptova

Kreiranje savršenog prompta je često izazovan i vremenski zahtjevan zadatak, koji zahtijeva duboko razumijevanje ciljne domene i nijansi jezičkih modela. Tu nastupa tehnika "Destilacije promptova", nudeći moćan pristup inženjeringu promptova koji koristi mogućnosti velikih jezičkih modela (LLM) za pojednostavljenje i optimizaciju procesa.

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

Kroz iterativni proces generisanja, poboljšanja i integracije, Destilacija promptova vam omogućava da kreirate promptove koji su koherentniji, sveobuhvatniji i usklađeniji sa željenim zadatkom ili izlazom. Imajte na umu da se proces destilacije može obaviti ručno u jednom od mnogih "igrališta" koje nude veliki AI dobavljači kao što su OpenAI ili Anthropic, ili se može automatizovati kao dio koda vaše aplikacije, zavisno od slučaja upotrebe.

Kako funkcioniše

Destilacija promptova tipično uključuje sljedeće korake:

1. **Identifikacija osnovne namjere**: Analizirajte prompt kako biste utvrdili njegovu primarnu svrhu i željeni ishod. Uklonite sve suvišne informacije i fokusirajte se na osnovnu namjeru prompta.
2. **Eliminacija dvosmislenosti**: Pregledajte prompt u potrazi za dvosmislenim ili nejasnim jezikom. Razjasnite značenje i pružite specifične detalje koji će usmjeriti VI prema generisanju tačnih i relevantnih odgovora.

3. **Pojednostavljenje jezika**: Pojednostavite prompt koristeći jasan i koncizan jezik. Izbjegavajte složene rečenične strukture, žargon ili nepotrebne detalje koji mogu zbuniti VI ili unijeti šum.
4. **Pružanje relevantnog konteksta**: Uključite samo najrelevantnije kontekstualne informacije potrebne da VI efikasno razumije i obradi prompt. Izbjegavajte uključivanje irelevantnih ili redundantnih detalja koji mogu odvući pažnju od osnovne namjere.
5. **Iteracija i poboljšanje**: Kontinuirano iterirajte i poboljšavajte prompt na osnovu VI odgovora i povratnih informacija. Evaluirajte generisane izlaze i napravite potrebna prilagođavanja kako biste poboljšali jasnoću i efikasnost prompta. Opcionalno verzionišite svoje promptove u bazi podataka koristeći Prompt Object kako biste pratili iteracije i dali sebi mogućnost da lako poništite promjene tokom izvršavanja.

Generisanje početnog prompta

Prvi korak u destilaciji promptova je da se VJM-u (velikom jezičkom modelu) pruži opis visokog nivoa ili namjera željenog zadatka ili izlaza. Ovo može biti kratka izjava ili skup ključnih riječi koje obuhvataju suštinu onoga što bi prompt trebao postići. VJM tada generiše početni prompt na osnovu ovog unosa, uključujući relevantna uputstva, primjere ili kontekst koje smatra prikladnim.

Pokušajmo dati Claude 3 Opus sljedeći početni prompt:

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

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

Claude odgovara sa sljedećim izlazom:

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

When interacting with clone owners, you should:

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

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

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

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

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

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

Usavršavanje uputa

Nakon što se generiše početna uputa, pregledajte je i zatim zatražite od VJM-a da razmotri bilo koji dodatni kontekst, ograničenja ili povratne informacije. Pustite VJM da generiše usavršenu verziju upute, uzimajući u obzir dobijeni unos i njegovo vlastito razumijevanje zadatka. Ovaj proces usavršavanja može se ponavljati iterativno, gdje svaka iteracija dodatno usavršava uputu na osnovu saradnje između korisnika i VJM-a.

Evo mog zahtjeva za usavršavanje, nastavljajući prethodni proces destilacije.

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

I Claudeov odgovor:

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

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

When interacting with clone owners, you should:

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

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

3. Maintain a friendly, professional, and client-focused tone in all
```

```
interactions, prioritizing customer satisfaction and working diligently to
understand and meet their needs.

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

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

Kompresija promptova

Kao što ste možda primijetili, JJM-ovi znaju biti opširni, osim ako ih ne zamolite da budu sažeti. Prompt na kojem smo radili u prethodnim sekcijama je prilično dugačak, pa hajde da zamolimo Claude da ga smanji:

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

Claude odgovara, sa 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

Pored usavršavanja samog prompta, VJM također može generirati odgovarajuće sistemske direktive ili kontekstualne informacije za usmjeravanje konačnog izlaza. Kada radite inženjering promptova za AI rutine koje će biti integrirane u vaš aplikacijski kod, u ovoj fazi destilacije ćete se gotovo sigurno fokusirati na ograničenja izlaza, ali možete raditi i na željenom tonu, stilu, formatu ili bilo kojim drugim relevantnim parametrima koji utječu na generirani odgovor.

Finalno sastavljanje prompta

Vrhunac procesa Destilacije promptova je sastavljanje finalnog prompta. To uključuje kombiniranje usavršenog prompta, generiranih sistemskih direktiva i integriranog konteksta u kohezivan i sveobuhvatan kod koji je spreman za korištenje za generiranje željenog izlaza.

Možete eksperimentirati s kompresijom promptova ponovno u fazi finalnog sastavljanja prompta, tražeći od VJM-a da smanji formulaciju prompta na najkraći mogući niz tokena, zadržavajući pritom suštinu njegovog ponašanja. To je svakako vježba s neizvjesnim ishodom, ali posebno u slučaju promptova koji će se izvršavati u većem obimu, dobici u efikasnosti mogu vam uštedjeti dosta novca u potrošnji tokena.

Ključne prednosti

Korištenjem znanja i generativnih sposobnosti VJM-ova za usavršavanje vaših promptova, vaši rezultirajući promptovi će vjerojatnije biti dobro strukturirani, informativni i prilagođeni specifičnom zadatku. Iterativni proces usavršavanja pomaže osigurati da su promptovi visoke kvalitete i efektivno capture željenu namjeru. Ostale prednosti uključuju:

Efikasnost i brzina: Destilacija promptova pojednostavljuje proces inženjeringa promptova automatiziranjem određenih aspekata kreiranja i usavršavanja promptova. Kolaborativna priroda tehnike omogućava brže približavanje efektivnom promptu, smanjujući vrijeme i trud potreban za ručno kreiranje promptova.

Konzistentnost i skalabilnost: Korištenje VJM-ova u procesu inženjeringa promptova pomaže održavanju konzistentnosti među promptovima, jer VJM-ovi mogu učiti i primjenjivati najbolje prakse i obrasce iz prethodnih uspješnih promptova. Ova konzis-

tentnost, u kombinaciji sa sposobnošću generiranja promptova u većem obimu, čini Destilaciju promptova vrijednom tehnikom za AI-pogođene aplikacije velikih razmjera.

Projektna ideja: Alati na nivou biblioteke koji pojednostavljuju proces verzioniranja i ocjenjivanja promptova u sistemima koji rade automatske destilacije promptova kao dio njihovog aplikacijskog koda.

Za implementaciju Destilacije promptova, developeri mogu dizajnirati tok rada ili pipeline koji integrira VJM-ove u različitim fazama procesa inženjeringa promptova. Ovo se može postići kroz API pozive, prilagođene alate ili integrisana razvojna okruženja koja omogućavaju nesmetanu interakciju između korisnika i VJM-ova tokom kreiranja promptova. Specifični detalji implementacije mogu varirati ovisno o odabranoj VJM platformi i zahtjevima aplikacije.

Šta je sa finim podešavanjem?

U ovoj knjizi, detaljno obrađujemo inženjering promptova i GPD, 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.

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

Slično tome, Generiranje potpomognuto dohvatom (GPD) nudi moćan pristup integraciji AI-ja u aplikacije. Dinamičkim dohvatanjem relevantnih informacija iz vanjskih baza znanja ili dokumenata, GPD pruža modelu fokusirani kontekst u vrijeme promptovanja. Ovo omogućava 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 duboki nivo prilagođavanja, često dolazi sa značajnim računarskim troškovima, zahtjevima za podacima i režijskim troškovima održavanja. Za većinu scenarija razvoja aplikacija, kombinacija efektivnog inženjeringa promptova i GPD-a bi trebala biti dovoljna za postizanje željene AI-pogođene funkcionalnosti i korisničkog iskustva.

Generisanje potpomognuto pronalaženjem (RAG)

Šta je Generisanje potpomognuto pronalaženjem?

Kako RAG funkcioniše?

Zašto koristiti RAG u vašim aplikacijama?

Implementacija RAG-a u Vašoj Aplikaciji

Priprema Izvora Znanja (Segmentacija)

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Segmentacija na propozicije

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Napomene o implementaciji

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Provjera kvalitete

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Prednosti preuzimanja zasnovanog na propozicijama

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Primjeri RAG-a iz stvarnog svijeta

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Studija slučaja: RAG u aplikaciji za pripremu poreza bez ugrađivanja

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Inteligentna Optimizacija Upita (IOU)

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Ponovno Rangiranje

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

RAG Procjena (RAGAs)

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Vjernost

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Relevantnost odgovora

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Preciznost konteksta

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Relevantnost konteksta

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Odziv konteksta

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Odziv entiteta konteksta

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Semantička sličnost odgovora (ANSS)

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Tačnost odgovora

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Kritika aspekata

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Izazovi i buduće perspektive

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Semantička segmentacija: Unapređenje preuzimanja sa segmentacijom svjesnom konteksta

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Hijerarhijsko indeksiranje: Strukturiranje podataka za poboljšano pronalaženje

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Self-RAG: Samoreflektivno poboljšanje

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

HyDE: Hipotetička ugrađivanja dokumenata

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Šta je kontrastno učenje?

Mnoštvo radnika

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

Umjesto da se oslanjaju isključivo na složene algoritme ili vremenski zahtjevne ručne implementacije, programeri mogu konceptualizirati AI komponente kao inteligentne, posvećene, ljudima slične entitete koji se mogu pozvati kad god je potrebno da riješe složene probleme i pruže rješenja zasnovana na njihovom treningu i znanju. Ovi entiteti se ne mogu dekoncentrisati niti uzeti bolovanje. Oni ne odlučuju spontano da rade stvari na drugačiji način od onog kako im je naloženo, i općenito govoreći, ako su pravilno programirani, ne prave ni greške.

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 određeni aspekt problema, donoseći svoje jedinstvene vještine i mogućnosti. Distribucijom radnog opterećenja među više AI radnika, aplikacija može postići veću efikasnost, skalabilnost i prilagodljivost.

Na primjer, razmotrite web aplikaciju koja zahtijeva moderaciju korisnički generisanog sadržaja u stvarnom vremenu. Implementacija sveobuhvatnog sistema za moderaciju 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 komponente za višekratnu upotrebu

Ključni aspekt pristupa Mnoštva radnika je njegova modularnost. Zagovornici objektno-orijentiranog programiranja nam već decenijama govore da o interakcijama objekata razmišljamo kao o porukama. Pa, AI radnici mogu biti dizajnirani kao nezavisne komponente za višekratnu upotrebu koje mogu "razgovarati jedni s drugima" putem običnog jezika, gotovo kao da su stvarno mali ljudi koji razgovaraju međusobno. Ovaj labavo povezani pristup omogućava aplikaciji da se prilagođava i razvija tokom vremena, kako se pojavljuju nove AI tehnologije ili se mijenjaju zahtjevi poslovne logike.

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

protive korištenju njihovih privatnih podataka za treniranje novih AI modela. Jeste li provjerili nivo privatnosti koji pruža pružalac modela koji koristite?

AI radnici kao mikroservisi?

Dok čitate o pristupu Mnoštva radnika, možda ćete primijetiti neke sličnosti s arhitekturom mikroservisa. Oba naglašavaju razlaganje složenih sistema u manje, upravljivije i nezavisno primjenjive 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 u svojim zadacima i međusobno komuniciraju kroz jasne interfejse i komunikacijske protokole.

Međutim, postoje neke ključne razlike koje treba imati na umu. Dok se mikroservisi tipično implementiraju kao odvojeni procesi ili servisi koji se izvršavaju na različitim mašinama ili kontejnerima, AI radnici se mogu implementirati kao samostalne komponente unutar jedne aplikacije ili kao odvojeni servisi, ovisno o vašim specifičnim zahtjevima i potrebama za skalabilnošću. Dodatno, komunikacija između AI radnika često uključuje razmjenu bogatih informacija baziranih na prirodnom jeziku, kao što su promptovi, instrukcije i generirani sadržaj, umjesto strukturiranijih formata podataka koji se obično koriste u mikroservisima.

Uprkos ovim razlikama, principi modularnosti, labave povezanosti i jasnih komunikacijskih interfejsa ostaju centralni za oba obrasca. Primjenjujući ove principe na vašu arhitekturu AI radnika, možete kreirati fleksibilne, skalabilne i održive sisteme 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

Praktično 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 promjene vezanih za korisničke račune.

Njena direktiva glasi ovako:

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

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

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

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

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

Raspon funkcija dostupnih `AccountManager`-u daje mu mogućnost uređivanja korisničke pretplate, dodavanja i uklanjanja AI konsultanata i drugih vrsta plaćenih dodataka, te slanja obavještenja putem e-maila vlasniku računa. Pored funkcije `finished`,

također može `notify_human_administrator` ako naiđe na grešku tokom obrade ili zahtijeva bilo kakvu drugu vrstu pomoći sa zahtjevom.

Primijetite da u slučaju pitanja, `AccountManager` može odabrati pretraživanje Olympijine baze znanja, gdje može pronaći upute o tome kako postupati u graničnim slučajevima i bilo kojoj drugoj situaciji u kojoj nije siguran kako nastaviti.

Primjene u e-trgovini

U domenu 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 generisanje personaliziranih preporuka proizvoda. Analiziranjem ponašanja korisnika, historije kupovine i preferencija, ovi radnici mogu predložiti proizvode koji su prilagođeni interesima i potrebama svakog pojedinačnog korisnika.

Ključ za efikasne preporuke proizvoda je korištenje kombinacije kolaborativnog filtriranja i filtriranja zasnovanog na sadržaju. Kolaborativno filtriranje posmatra ponašanje sličnih korisnika kako bi identificiralo obrasce i dalo preporuke na osnovu onoga što su drugi sa sličnim ukusima kupili ili im se svidjelo. S druge strane, filtriranje zasnovano na sadržaju fokusira se na karakteristike i atribute samih proizvoda, preporučujući artikle koji dijele slične karakteristike s onima za koje je korisnik prethodno pokazao interes.

Evo pojednostavljenog primjera kako možete implementirati radnika za preporuke proizvoda u Ruby-ju, 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 je pod uticajem F# i Rust jezika. Više o tome možete pročitati u objašnjenju tehnike mog prijatelja Chad Wooley-a na objašnjenju tehnike na GitLab-u.

U ovom primjeru, `ProductRecommendationWorker` uzima korisnika kao ulaz i generiše personalizovane preporuke proizvoda proslijeđivanjem vrijednosnog objekta kroz lanac funkcionalnih koraka. Analizirajmo svaki korak:

1. `ValidateUser.validate`: Ovaj korak osigurava da je korisnik validan i podoban za personalizovane preporuke. Provjerava da li korisnik postoji, da li je aktivan i da li ima potrebne podatke za generisanje preporuka. Ako validacija ne uspije, vraća se rezultat greške i lanac se prekida.
2. `AnalyzeCurrentSession.analyze`: Ako je korisnik validan, ovaj korak analizira trenutnu sesiju pregledanja korisnika kako bi prikupio kontekstualne informacije. Posmatra nedavne interakcije korisnika, kao što su pregledani proizvodi,

upiti za pretragu i sadržaj korpe, kako bi razumio njihove trenutne interese i namjere.

3. `CollaborativeFilter.filter`: Koristeći *ponašanje sličnih korisnika*, ovaj korak primjenjuje tehnike kolaborativnog filtriranja za identifikaciju proizvoda koji bi mogli biti interesantni korisniku. Uzima u obzir faktore poput historije kupovine, ocjena i interakcija korisnik-proizvod kako bi generisao set kandidata za preporuke.
4. `ContentBasedFilter.filter`: Ovaj korak dalje usavršava kandidate za preporuke primjenjujući filtriranje zasnovano na sadržaju. Upoređuje atribute i karakteristike kandidovanih proizvoda sa *korisničkim preferencama i historijskim podacima* kako bi odabrao najrelevantnije stavke.
5. `ProductSelector.select`: Konačno, ovaj korak bira top N proizvoda iz filtriranih preporuka na osnovu predefinisanih kriterija, kao što su ocjena relevantnosti, popularnost ili druga poslovna pravila. Odabrani proizvodi se zatim vraćaju kao konačne personalizovane preporuke.

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

U `case` izrazu na kraju, radimo podudaranje uzoraka na konačnom rezultatu. Ako je rezultat greška (`ProductRecommendationError`), bilježimo grešku koristeći alat poput Honeybadger-a za praćenje i otklanjanje grešaka. Ako je rezultat uspješan (`ProductRecommendations`), emitujemo događaj `:new_recommendations` koristeći Wisper biblioteku za objavljivanje/pretplatu, prosljeđujući korisnika i generisane preporuke.

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

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

Naravno, ovo je pojednostavljen primjer, i u stvarnom scenariju, trebali biste se integrisati sa vašom e-commerce platformom, rukovati graničnim slučajevima, pa čak i upustiti se u implementaciju algoritama za preporuke. Međutim, osnovni principi razlaganja problema na manje korake i korištenja tehnika funkcionalnog programiranja ostaju isti.

Detekcija prevara

Evo pojednostavljenog primjera kako možete implementirati worker za detekciju prevara koristeći isti stil Programiranja orijentisanog na željezničke pruge (ROP) u Ruby-ju:

```ruby
class FraudDetectionWorker
  include Wisper::Publisher

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

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

```
  end
end
```

Klasa `FraudDetection` je *value object* koji enkapsulira stanje detekcije prevare za datu transakciju. Ona pruža strukturirani način za analizu i procjenu rizika od prevare povezane s transakcijom na osnovu različitih faktora rizika.

```
class FraudDetection
  RISK_THRESHOLD = 0.8

  attr_accessor :transaction, :risk_factors

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

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

  def high_risk?
    fraud_probability > RISK_THRESHOLD
  end

  private

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

Klasa `FraudDetection` ima sljedeće atribute:

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

- `risk_factors`: Niz koji pohranjuje faktore rizika povezane s transakcijom. Svaki faktor rizika je predstavljen kao heš, gdje je ključ opis faktora rizika, a vrijednost je vjerovatnoća prevare povezana s tim faktorom rizika.

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

Metoda `high_risk?` koja će biti provjerena na kraju lanca je predikatna metoda koja upoređuje `fraud_probability` (izračunatu sumiranjem vjerovatnoća svih faktora rizika) sa `RISK_THRESHOLD`.

Klasa `FraudDetection` pruža čist i enkapsuliran način upravljanja detekcijom prevare za transakciju. Omogućava dodavanje više faktora rizika, svaki sa svojim opisom i vjerovatnoćom, i pruža metodu za određivanje da li se transakcija smatra visokorizičnom na osnovu izračunate vjerovatnoće prevare. Klasa se može lako integrisati u veći sistem za detekciju prevara, gdje različite komponente mogu sarađivati u procjeni i ublažavanju rizika od lažnih transakcija.

Konačno, s obzirom da je ovo ipak knjiga o programiranju koristeći AI, evo primjera implementacije klase `CheckCustomerHistory` koja koristi AI obradu pomoću modula `ChatCompletion` iz moje Raix biblioteke:

```ruby
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` definiše konstantu `INSTRUCTION` koja pruža specifična uputstva AI modelu o tome kako da analizira historiju transakcija kupca za potencijalne indikatore prevare putem sistemske direktive

Metoda `self.check` je klasna metoda koja inicijalizira novu instancu `CheckCustomerHistory` sa `fraud_detection` objektom i poziva metodu `call` da izvrši analizu historije kupca.

Unutar metode `call`, historija transakcija kupca se preuzima i formatira u transkript koji se proslijeđuje AI modelu. AI model analizira historiju transakcija na osnovu datih instrukcija i vraća sažetak svojih nalaza.

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

Korištenjem modula `ChatCompletion`, klasa `CheckCustomerHistory` može iskoristiti snagu AI-ja za analizu historije transakcija kupca i identifikaciju potencijalnih indikatora prevare. Ovo omogućava sofisticiranije i adaptivnije tehnike detekcije prevare, jer AI model može učiti i prilagođavati se novim obrascima i anomalijama tokom vremena.

Ažurirani `FraudDetectionWorker` i klasa `CheckCustomerHistory` pokazuju kako se AI radnici mogu besprijekorno integrisati, unapređujući proces detekcije prevare sa inteligentnim mogućnostima analize i donošenja odluka.

Analiza Sentimenta Kupaca

Evo još jednog sličnog primjera kako možete implementirati radnika za analizu sentimenta kupaca. Ovaj put sa mnogo manje objašnjenja, jer biste već trebali shvatati kako ovaj stil programiranja funkcioniše:

```
class CustomerSentimentAnalysisWorker
  include Wisper::Publisher

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

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

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

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 osnovu zahtjeva pacijenta i dostupnosti zdravstvenog pružatelja usluga.

Prikupljanje medicinske historije: Tokom prijema pacijenata, AI radnici mogu pomoći u prikupljanju i dokumentovanju medicinske historije pacijenta. Mogu voditi interaktivne dijaloge s pacijentima, postavljajući relevantna pitanja o njihovim prethodnim medicinskim stanjima, lijekovima, alergijama i porodičnoj historiji. AI radnici mogu koristiti tehnike obrade prirodnog jezika za tumačenje i strukturiranje prikupljenih informacija, osiguravajući da su tačno zabilježene u elektronskom 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 faktorima. Koristeći medicinske baze znanja i modele mašinskog učenja, ovi radnici mogu analizirati pružene informacije i generirati preliminarne diferencijalne dijagnoze ili preporučiti odgovarajuće sljedeće korake, kao što su zakazivanje konsultacija sa zdravstvenim radnikom ili predlaganje mjera samopomoći.

Verifikacija osiguranja: AI radnici mogu pomoći pri verifikaciji osiguranja tokom prijema pacijenata. Mogu prikupljati podatke o osiguranju pacijenta, komunicirati s osiguravajućim društvima putem API-ja ili web servisa, te provjeravati pravo na pokriće i beneficije. Ova automatizacija pomaže u pojednostavljenju procesa verifikacije osiguranja, smanjujući administrativno opterećenje i osiguravajući tačno prikupljanje informacija.

Edukacija pacijenata i upute: AI radnici mogu pružiti pacijentima relevantne edukativne materijale i upute na osnovu njihovih specifičnih medicinskih stanja ili predstojećih procedura. Mogu isporučiti personalizirani sadržaj, odgovarati na česta pitanja i pružati smjernice o pripremama prije termina, uputama za uzimanje lijekova ili njezi nakon tretmana. Ovo pomaže u održavanju informiranosti i angažmana pacijenata tokom njihovog zdravstvenog putovanja.

Korištenjem AI radnika u prijemu pacijenata, zdravstvene organizacije mogu poboljšati efikasnost, smanjiti vrijeme čekanja i unaprijediti cjelokupno iskustvo pacijenata. Ovi radnici mogu obavljati rutinske zadatke, prikupljati tačne informacije i pružati personaliziranu pomoć, omogućavajući zdravstvenim radnicima da se fokusiraju na pružanje kvalitetne njege 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 elektronski zdravstveni kartoni (EZK), medicinske slike, laboratorijski rezultati, nosivi uređaji i socijalne determinante zdravlja. Objedinjavanjem ovih informacija u sveobuhvatni profil pacijenta, AI radnici mogu pružiti holistički pregled zdravstvenog stanja pacijenta i faktora rizika.

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

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

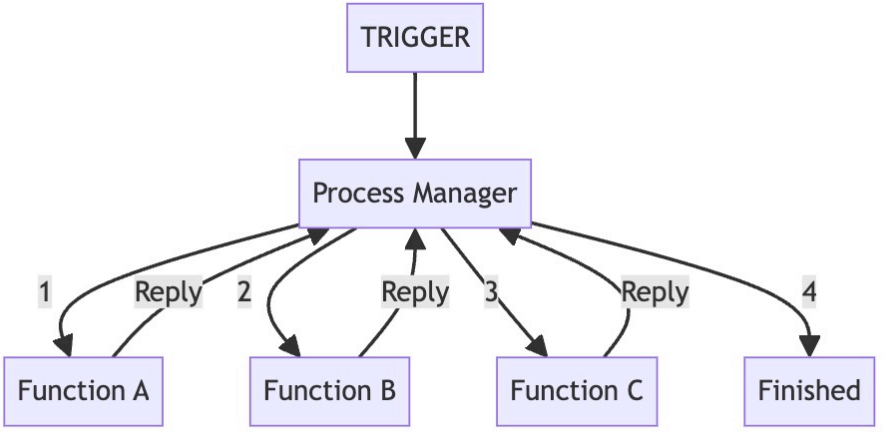
Personalizirani profili rizika: AI radnici mogu generirati personalizirane profile rizika za svakog pacijenta, ističući specifične faktore koji doprinose njihovim ocjenama rizika. Ovi profili mogu uključivati uvide u način života pacijenta, genetske predispozicije, faktore okoline i socijalne determinante zdravlja. Pružanjem detaljnog pregleda faktora rizika, AI radnici mogu pomoći zdravstvenim radnicima da prilagode strategije prevencije i planove liječenja individualnim potrebama pacijenata.

Kontinuirano praćenje rizika: AI radnici mogu kontinuirano pratiti podatke o pacijentima i ažurirati procjene rizika u realnom vremenu. Kako nove informacije postaju dostupne, kao što su promjene u vitalnim znakovima, laboratorijskim rezultatima ili pridržavanju terapije, AI radnici mogu ponovno izračunati ocjene rizika i upozoriti zdravstvene radnike na značajne promjene. Ovo proaktivno praćenje omogućava pravovremene intervencije i prilagođavanje planova njege pacijenata.

Podrška kliničkom odlučivanju: AI radnici mogu integrirati rezultate procjene rizika u sisteme za podršku kliničkom odlučivanju, pružajući zdravstvenim radnicima preporuke i upozorenja zasnovana na dokazima. Na primjer, ako ocjena rizika pacijenta za određeno stanje prelazi određeni prag, AI radnik može potaknuti zdravstvenog radnika da razmotri specifične dijagnostičke testove, preventivne mjere ili opcije liječenja na osnovu kliničkih smjernica i najboljih praksi.

Ovi radnici mogu obrađivati ogromne količine podataka o pacijentima, primjenjivati sofisticirane analize i generisati praktične uvide koji podržavaju kliničko odlučivanje. Ovo u konačnici dovodi do poboljšanih ishoda za pacijente, smanjenih troškova zdravstvene zaštite i unaprijeđenog upravljanja zdravljem populacije.

AI Radnik kao Upravitelj Procesa



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

Kada AI radnik djeluje kao Upravitelj Procesa, prima dolaznu poruku koja inicijalizira proces, poznatu kao *poruka okidača*. AI radnik zatim održava stanje izvršenja procesa (kao transkript razgovora) i upravlja porukom kroz niz koraka obrade implementiranih kao funkcije alata, koji mogu biti sekvencijalni ili paralelni, i pozivaju se po njegovom nahođenju.

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

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

Pohranite Vaše Poruke Okidača

Prema mom iskustvu, pametno je implementirati vašu poruku okidača kao objekat podržan bazom podataka. Na taj način je svaka instanca procesa identificirana jedinstvenim primarnim ključem i daje vam mjesto za pohranu stanja povezanog s izvršavanjem, uključujući AI transkript razgovora.

Na primjer, evo pojednostavljene verzije Olympia-inog modela `AccountChange`, koji predstavlja zahtjev za promjenu 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 okidača koja pokreće proces za rukovanje zahtjevom za promjenu računa. Primijetite kako se emituje prema Olympia-inom Wisper sistemu za objavljivanje/pretplatu nakon što se transakcija kreiranja završi.

Pohranjivanje 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 omogućava jednostavnu identifikaciju i praćenje pojedinačnih zahtjeva. Ovo je posebno korisno za potrebe revizijskog bilježenja, jer omogućava sistemu da održava historijski zapis svih promjena računa, uključujući kada su zatražene, koje promjene su zatražene i trenutno stanje svakog zahtjeva.

U datom primjeru, klasa `AccountChange` uključuje polja kao što su `description` za bilježenje detalja zatražene promjene, `state` za predstavljanje trenutnog stanja zahtjeva (npr. zatraženo, završeno, zahtijeva_ljudski_pregled), i `transcript` za pohranjivanje transkripta AI konverzacije vezane za zahtjev. Polje `description` je stvarni prompt koji se koristi za pokretanje prvog chat completion-a sa AI-em. Pohranjivanje ovih podataka pruža vrijedan kontekst i omogućava bolje praćenje i analizu procesa promjene računa.

Pohranjivanje poruka okidača u bazu podataka omogućava robusno rukovanje greškama i oporavak. Ako se greška dogodi tokom obrade zahtjeva za promjenu računa, sistem

označava zahtjev kao neuspješan i prebacuje ga u stanje koje zahtijeva ljudsku intervenciju. Ovo osigurava da nijedan zahtjev nije izgubljen ili zaboravljen, i da se svi problemi mogu pravilno adresirati i riješiti.

AI radnik, kao Upravitelj Procesa, pruža centralnu tačku kontrole i omogućava snažne mogućnosti izvještavanja i otklanjanja grešaka u procesu. Međutim, važno je napomenuti da korištenje AI radnika kao Upravitelja Procesa za svaki scenarij toka rada u vašoj aplikaciji može biti pretjerano.

Integracija AI Radnika U Arhitekturu Vaše Aplikacije

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

Dizajniranje Jasnih Interfejsa i Komunikacijskih Protokola

Za omogućavanje besprijekorne integracije između AI radnika i drugih komponenti aplikacije, ključno je definisati jasne interfejse i komunikacijske protokole. Razmotrite sljedeće pristupe:

Integracija zasnovana na API-ju: Izložite funkcionalnost AI radnika kroz dobro definisane API-je, kao što su RESTful endpoint-i ili GraphQL sheme. Ovo omogućava drugim komponentama da komuniciraju sa AI radnicima koristeći standardne HTTP zahtjeve i odgovore. Integracija zasnovana na API-ju pruža jasan ugovor između

AI radnika i komponenti koje ih koriste, olakšavajući razvoj, testiranje i održavanje integracijskih tačaka.

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

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

Rukovanje Protokom Podataka i Sinhronizacija

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

Priprema podataka: Prije slanja podataka AI radnicima, možda ćete morati izvršiti razne zadatke pripreme podataka, kao što su čišćenje, formatiranje i/ili transformacija ulaznih podataka. Ne želite samo osigurati da AI radnici mogu efikasno obrađivati podatke, već i da ne trošite tokene dajući pažnju informacijama koje bi radnik mogao smatrati beskorisnim u najboljem slučaju, a ometajućim u najgorem. Priprema podataka

može uključivati zadatke poput uklanjanja šuma, rukovanja nedostajućim vrijednostima ili konverzije tipova podataka.

Postojanost podataka: Kako ćete pohraniti i održavati podatke koji teku u i iz AI radnika? Razmotrite faktore kao što su volumen podataka, obrasci upita i skalabilnost. Da li trebate sačuvati transkript AI-a kao odraz njegovog "procesa razmišljanja" za potrebe revizije ili otklanjanja grešaka, ili je dovoljno imati zapis samo o rezultatima?

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

Sinhronizacija podataka: Kada više komponenti, uključujući AI radnike, pristupa i modificira zajedničke podatke, važno je implementirati odgovarajuće mehanizme sinhronizacije kako bi se održala konzistentnost podataka. Strategije zaključavanja baze podataka, kao što su optimističko ili pesimističko zaključavanje, mogu vam pomoći da spriječite konflikte i osigurate integritet podataka. Implementirajte tehnike upravljanja transakcijama za grupiranje povezanih operacija s podacima i održavanje svojstava atomnosti, konzistentnosti, izolacije i trajnosti (ACID)

Upravljanje greškama i oporavak: Implementirajte robusne mehanizme za upravljanje greškama i oporavak kako biste se nosili s problemima vezanim za podatke koji se mogu pojaviti tokom procesa toka podataka. Elegantno rukujte izuzecima i pružite smislene poruke o greškama koje pomažu pri debugiranju. Implementirajte mehanizme ponovnih pokušaja i rezervne strategije za rukovanje privremenim kvarovima ili prekidima mreže. Definirajte jasne procedure za oporavak i vraćanje podataka u slučaju oštećenja ili gubitka podataka.

Pažljivim dizajniranjem i implementacijom mehanizama toka i sinhronizacije podataka, možete osigurati da vaši AI radnici imaju pristup tačnim, konzistentnim i ažurnim

podacima. Ovo im omogućava da efikasno obavljaju svoje zadatke i proizvode pouzdane rezultate.

Upravljanje životnim ciklusom AI radnika

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

Implementirajte sveobuhvatne mehanizme praćenja i bilježenja za praćenje zdravlja i performansi AI radnika. Prikupljajte metrike kao što su iskorištenost resursa, vrijeme obrade, stope grešaka i propusnost. Koristite centralizovane sisteme za bilježenje kao što je ELK stack (Elasticsearch, Logstash, Kibana) za agregaciju i analizu zapisa iz više AI radnika.

Ugradite toleranciju na greške i otpornost u arhitekturu AI radnika. Implementirajte mehanizme za rukovanje greškama i oporavak kako biste elegantno rukovali kvarovima ili izuzecima. Veliki jezički modeli su još uvijek tehnologija u razvoju; pružatelji usluga često prestaju s radom u neočekivanim trenucima. Koristite mehanizme ponovnih pokušaja i prekidače strujnog kola da spriječite kaskadne greške.

Komponibilnost i orkestracija AI radnika

Jedna od ključnih prednosti arhitekture AI radnika je njena komponibilnost, koja vam omogućava da kombinirate i orkestrite više AI radnika za rješavanje složenih problema. Razbijanjem većeg zadatka na manje, lakše upravljive podzadatke, kojima upravlja specijalizovani AI radnik, možete stvoriti moćne i fleksibilne sisteme. U ovom odjeljku, istražit ćemo različite pristupe komponovanju i orkestraciji “mnoštva” AI radnika.

Ulančavanje AI radnika za višekoračne tokove rada

U mnogim scenarijima, složeni zadatak se može rastaviti na niz sekvencijalnih koraka, gdje izlaz jednog AI radnika postaje ulaz za sljedećeg. Ovo ulančavanje AI radnika stvara višekoračni tok rada ili cjevovod. Svaki AI radnik u lancu fokusira se na specifični podzadatak, a konačni izlaz je rezultat kombinovanih napora svih radnika.

Razmotrimo primjer u kontekstu Ruby on Rails aplikacije za obradu korisnički generisanog sadržaja. Tok rada uključuje sljedeće korake, koji su, priznajemo, vjerojatno svaki previše jednostavan da bi bio vrijedan razlaganja na ovaj način u stvarnim slučajevima upotrebe, ali oni 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. **Detekcija jezika**: AI radnik koji identificira jezik očišćenog teksta.

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

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

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

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

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

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

U ovom primjeru, klasa `ContentProcessor` se inicijalizira 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, detektovani 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 tok rada njihovom paralelnom obradom.

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

Za postizanje paralelne obrade u Ruby-ju, možete koristiti tehnike konkurentnosti kao što su niti ili asinhorno programiranje. Evo primjera kako možete modificirati klasu `ContentProcessor` da obrađuje završna tri radnika paralelno koristeći niti:

```ruby
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 kreiranje `Concurrent::Future` objekata za svakog od nezavisnih AI radnika. „Future“ predstavlja računanje koje će se izvršiti asinhrono u zasebnoj niti.

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

Pozivanjem metode `value` na svakom `Future` objektu, čekamo da se računanje završi

i dohvaćamo rezultat. Metoda `value` blokira izvršavanje dok rezultat nije dostupan, osiguravajući da su svi paralelni radnici završili obradu prije nastavka.

Na kraju, konstruišemo izlazni hash sa očišćenim tekstom i rezultatima iz paralelnih radnika, baš kao u originalnom primjeru.

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

Međutim, važno je napomenuti da stvarni dobici u performansama zavise od različitih faktora, kao što su složenost svakog radnika, dostupni sistemski resursi i režijski troškovi upravljanja nitima. Uvijek je dobra praksa mjeriti performanse i profilirati svoj kod kako biste odredili optimalni nivo paralelizma za vaš specifični slučaj upotrebe.

Dodatno, kada implementirate paralelnu obradu, vodite računa o svim dijeljenim resursima ili zavisnostima između radnika. Osigurajte da radnici mogu raditi nezavisno bez konflikata ili uvjeta utrke. Ako postoje zavisnosti ili dijeljeni resursi, možda ćete trebati implementirati odgovarajuće mehanizme sinhronizacije kako biste održali integritet podataka i izbjegli probleme poput potpunih blokada ili nekonzistentnih rezultata.

Ruby-jeva Globalna brava interpretera i Asinhrona obrada

Važno je razumjeti implikacije Ruby-jeve Globalne brave interpretera (GIL) kada razmatramo asinhronu obradu zasnovanu na nitima u Ruby-ju.

GIL je mehanizam u Ruby-jevom interpreteru koji osigurava da samo jedna nit može izvršavati Ruby kod u jednom trenutku, čak i na procesorima sa više jezgara. To znači

da iako se više niti može kreirati i njima upravljati unutar Ruby procesa, samo jedna nit može aktivno izvršavati Ruby kod u bilo kojem trenutku.

GIL je dizajniran da pojednostavi implementaciju Ruby interpretera i pruži sigurnost niti za Ruby-jeve interne strukture podataka. Međutim, također ograničava potencijal za istinsko paralelno izvršavanje Ruby koda.

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

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

To znači da je asinhrona obrada zasnovana na nitima u Ruby-ju najefikasnija za U/I-vezane zadatke, kao što je čekanje na odgovore vanjskih API-ja (poput LLM modela hostanih od treće strane) ili izvođenje U/I operacija nad datotekama. Kada nit naiđe na U/I operaciju, može osloboditi GIL, omogućavajući drugim nitima da se izvršavaju dok čekaju da se U/I operacija završi.

S druge strane, za CPU-vezane zadatke, kao što su intenzivna računanja ili dugotrajna AI obrada, GIL može ograničiti potencijalne dobitke u performansama kod paralelizma zasnovanog na nitima. Budući da samo jedna nit može izvršavati Ruby kod u jednom trenutku, ukupno vrijeme izvršavanja možda neće biti značajno smanjeno u poređenju sa sekvencijalnom obradom.

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

- Korištenje paralelizma zasnovanog na procesima sa više Ruby procesa, od kojih svaki radi na zasebnom CPU jezgru.

- Korištenje vanjskih biblioteka ili okvira koji pružaju native ekstenzije ili interfejse prema jezicima bez GIL-a, kao što su C ili Rust.,
- Korištenje distribuiranih računarskih okvira ili redova poruka za distribuciju zadataka preko više mašina ili procesa.

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

Ensemble tehnike za poboljšanu preciznost

Ensemble tehnike uključuju kombinovanje izlaza više AI radnika kako bi se poboljšala ukupna preciznost ili robusnost sistema. Umjesto oslanjanja na jednog AI radnika, ensemble tehnike koriste kolektivnu inteligenciju više radnika za donošenje informisanijih odluka.

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

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

Razmotrimo primjer gdje imamo tri AI radnika za analizu sentimenta, od kojih svaki koristi različiti model ili je opremljen različitim kontekstima. Možemo kombinovati njihove izlaze koristeći većinsko glasanje kako bismo odredili konačnu predikciju sentimenta.

```
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`. se inicijalizira sa tekstom i poziva tri različita AI radnika za analizu sentimenta. Metoda `analyze` prikuplja predviđanja od svakog radnika i određuje većinski sentiment koristeći metode `group_by` i `max_by`. Konačni rezultat je sentiment koji dobija najviše glasova od ansambla radnika

Ansambli su očigledno slučaj gdje eksperimentisanje sa paralelizmom može biti vrijedno vašeg vremena.

Dinamički odabir i pozivanje AI radnika

U nekim, ako ne i većini slučajeva, odabir specifičnog AI radnika može zavisiti od uslova izvršavanja ili korisničkih unosa. Dinamički odabir i pozivanje AI radnika omogućavaju fleksibilnost i prilagodljivost sistema.

Možda ćete biti u iskušenju da pokušate uklopiti mnogo funkcionalnosti u jednog AI radnika, dajući mu mnogo funkcija i veliki komplikovani prompt koji objašnjava kako ih pozvati. Oduprite se iskušenju, vjerujte mi. Jedan od razloga zašto se pristup o kojem razgovaramo u ovom poglavlju zove "Mnoštvo radnika" je da nas podsjeti da je poželjno imati mnogo 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 osnovu 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 `process_query` akciju. Prvo koristi `QueryClassifierWorker` da odredi tip upita. Na osnovu klasificiranog tipa upita, kontroler dinamički odabire odgovarajućeg AI radnika za generisanje odgovora. Ovaj dinamički odabir omogućava chatbotu da obrađuje različite vrste upita i usmjerava ih ka relevantnim AI radnicima.

S obzirom da je rad `QueryClassifierWorker`-a relativno jednostavan i ne zahtijeva mnogo konteksta ili definicija funkcija, vjerovatno ga možete implementirati koristeći ultra-brzi mali LLM poput `mistralai/mixtral-8x7b-instruct:nitro`. Njegove sposobnosti su blizu GPT-4 nivoa na mnogim zadacima i, u vrijeme pisanja ovog teksta, Groq ga može posluživati nevjerovatnom brzinom od 444 tokena u sekundi.

Kombinovanje tradicionalne OPJ sa VJM-ovima

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

Razmišljajte o VJM-ovima kao o švajcarskim nožićima OPJ-a—nevjerovatno svestrani i moćni, ali ne nužno najbolji alat za svaki posao. Ponekad, namjenski alat poput vadičepa ili otvarača za konzerve može biti efikasniji za određeni zadatak. Slično tome, tradicionalne OPJ tehnike, kao što su grupisanje dokumenata, identifikacija tema i klasifikacija, često mogu pružiti ciljanije i isplativije rješenje za određene aspekte vašeg OPJ procesa.

Jedna od ključnih prednosti tradicionalnih OPJ tehnika je njihova računarska efikasnost. Ove metode, koje se često oslanjaju na jednostavnije statističke modele ili pristupe

zasnovane na pravilima, mogu obrađivati velike količine tekstualnih podataka mnogo brže i sa manjim računarskim opterećenjem u poređenju sa VJM-ovima. To ih čini posebno pogodnim za zadatke koji uključuju analizu i organizaciju velikih korpusa dokumenata, kao što su grupisanje sličnih članaka ili identifikacija ključnih tema unutar kolekcije tekstova.

Štaviše, tradicionalne OPJ tehnike često mogu postići visoku tačnost i preciznost za specifične zadatke, posebno kada su trenirane na domenski specifičnim skupovima podataka. Na primjer, dobro podešen klasifikator dokumenata koji koristi tradicionalne algoritme mašinskog učenja poput Mašina potpornih vektora (MPV) ili Naivnog Bayesa može precizno kategorizirati dokumente u predefinisane kategorije uz minimalne računarske troškove.

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

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

Na primjer, razmotrimo slučaj upotrebe gdje želite generisati izvještaj o trendovima za specifičnu domenu na osnovu velikog korpusa pojedinačnih dokumenata o trendovima. Umjesto da se oslanjate isključivo na VJM-ove, što može biti računarski skupo i vremenski zahtjevno za obradu velikih količina teksta, možete primijeniti hibridni

pristup:

1. Koristite tradicionalne OPJ tehnike, kao što su modeliranje tema (npr. Latentna Dirihleova alokacija) ili algoritmi grupisanja (npr. K-means), za grupisanje sličnih dokumenata o trendovima i identifikaciju ključnih tema unutar korpusa.
2. Proslijedite grupisane dokumente i identifikovane teme VJM-u, koristeći njegove superiorne sposobnosti razumijevanja i generisanja jezika za kreiranje koherentnih i informativnih sažetaka za svaku grupu ili temu.
3. Na kraju, koristite VJM za generisanje sveobuhvatnog izvještaja o trendovima kombinovanjem pojedinačnih sažetaka, isticanjem najznačajnijih trendova i pružanjem uvida i preporuka na osnovu objedinjenih informacija.

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

Dok se upuštate u svoje NLP (obrada prirodnog jezika) projekte, ključno je pažljivo procijeniti specifične zahtjeve i ograničenja svakog zadatka te razmotriti kako se tradicionalne NLP metode i LLMs (veliki jezički modeli) mogu zajedno iskoristiti za postizanje najboljih rezultata. Kombinovanjem efikasnosti i preciznosti tradicionalnih tehnika sa svestranošću i snagom LLM-ova, možete kreirati visoko efektivna i ekonomična NLP rješenja koja donose vrijednost vašim korisnicima i zainteresovanim stranama.

Upotreba alata

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

Šta je upotreba alata?

Upotreba alata, također poznata kao pozivanje funkcija, je tehnika koja omogućava programerima da definišu listu funkcija s kojima VJM može interagovati tokom procesa generisanja. Ovi alati mogu varirati od jednostavnih pomoćnih funkcija do kompleksnih

API-ja ili upita baza podataka. Omogućavajući VJM-u pristup 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 upotrebe alata je dati VJM-u mogućnost da dinamički odabere i izvrši odgovarajuće alate na osnovu korisničkog unosa ili zadatka koji treba obaviti. Umjesto da se oslanja isključivo na prethodno obučeno znanje modela, upotreba alata omogućava VJM-u da koristi vanjske resurse za generisanje preciznijih, relevantnijih i upotrebljivijih odgovora. Upotreba alata čini tehnike poput RAG-a (Retrieval Augmented Generation)

mnogo lakšim za implementaciju nego što bi to inače bile.

> Imajte na umu da, osim ako nije drugačije navedeno, ova knjiga pretpostavlja da vaš AI model nema pristup ugrađenim alatima na strani servera. Sve alate koje želite učiniti dostupnim vašem AI-u morate eksplicitno deklarisati u svakom API zahtjevu, uz odredbe za njihovo izvršavanje ako i kada vaš AI kaže da želi koristiti taj alat u svom odgovoru.

Potencijal upotrebe alata

Upotreba alata otvara širok spektar mogućnosti za aplikacije pokretane AI-em. Evo nekoliko primjera šta se može postići upotrebom alata:

1. **Četbotovi i virtuelni asistenti**: Povezivanjem VJM-a sa vanjskim alatima, četbotovi i virtuelni asistenti mogu obavljati složenije zadatke, kao što su preuzimanje informacija iz baza podataka, izvršavanje API poziva ili interakcija sa drugim sistemima. Na primjer, četbot bi mogao koristiti CRM alat za promjenu statusa prodaje na osnovu zahtjeva korisnika.
2. **Analiza podataka i uvidi**: VJM-ovi se mogu povezati sa alatima ili bibliotekama za analizu podataka kako bi izvršavali napredne zadatke obrade podataka. Ovo omogućava aplikacijama da generišu uvide, provode komparativne analize ili pružaju preporuke zasnovane na podacima na osnovu korisničkih upita.
3. **Pretraga i pronalaženje informacija**: Upotreba alata omogućava VJM-ovima interakciju sa pretraživačima, vektorskim bazama podataka ili drugim sistemima za pronalaženje informacija. Transformisanjem korisničkih upita u upite za pretragu, VJM može preuzeti relevantne informacije iz više izvora i pružiti sveobuhvatne odgovore na korisnička pitanja.

4. **Integracija sa vanjskim servisima**: Upotreba alata omogućava besprijekornu integraciju između aplikacija pokretanih AI-em i vanjskih servisa ili API-ja. Na primjer, VJM bi mogao komunicirati sa API-jem za vremenske prognoze kako bi pružio ažuriranja o vremenu u realnom vremenu ili sa API-jem za prevođenje kako bi generisao višejezične odgovore.

Tok rada upotrebe alata

Tok rada upotrebe alata tipično uključuje četiri ključna koraka:

1. Uključivanje definicija funkcija u kontekst vašeg zahtjeva
2. Dinamički (ili eksplicitni) odabir alata
3. Izvršavanje funkcije(a)
4. Opcionalni nastavak originalnog prompta

Hajde da detaljno pregledamo svaki od ovih koraka.

Uključivanje definicija funkcija u kontekst vašeg zahtjeva

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

Precizna sintaksa definicije alata je specifična za model.

Ovako definišete funkciju `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"]
    }
}
```

I evo kako biste definisali istu funkciju za GPT-4, proslijeđujući je kao vrijednost `tools` parametra:

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

```
    },
}
```

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

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

Pored stvarnih definicija 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 pomenute alate na raspolaganju:

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

Pružanje detaljnih opisa smatra se najvažnijim faktorom u performansama alata. Vaši opisi trebaju objasniti svaki detalj o alatu, uključujući:

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

Što više konteksta možete dati AI-u o svojim alatima, to će biti bolji u odlučivanju kada i kako ih koristiti. Na primjer, Anthropic preporučuje najmanje 3-4 rečenice po opisu alata za svoju Claude 3 seriju, više ako je alat složen.

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

Evo primjera Stripe-ovske API specifikacije funkcije:

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

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

U praksi, neki modeli imaju poteškoća sa ugniježđenim specifikacijama funkcija i sa složenim tipovima izlaznih podataka kao što su nizovi, rječnici itd. Ali 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 upotrebu i generira potrebne ulazne parametre za svaki alat.

U praksi, sposobnost AI-ja da pozove *tačno* odgovarajuću funkciju i *tačno* prati vašu specifikaciju za ulazne podatke je promjenjiva. Postavljanje hiperparametra temperature na 0.0 značajno pomaže, ali iz mog iskustva i dalje ćete povremeno dobijati greške. Te greške uključuju halucirane nazive funkcija, pogrešno imenovane ili jednostavno nedostajuće ulazne parametre. Parametri se prenose kao JSON, što znači da ćete ponekad vidjeti greške uzrokovane skraćenim, pogrešno navedenim ili na drugi način neispravnim JSON-om.

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

Prisilni (tj. Eksplicitni) odabir alata

Neki modeli vam daju mogućnost da prisilite pozivanje određene funkcije kao parametar u zahtjevu. U suprotnom, odluka o tome hoće li se funkcija pozvati ili ne je u potpunosti prepuštena AI-ju.

Mogućnost prisilnog pozivanja funkcije je ključna 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 predefiniranom toku rada koji zahtijeva izvršavanje određene funkcije u određeno vrijeme. Prisiljavanjem poziva možete garantovati da će željena funkcija biti pozvana umjesto da ljubazno molite AI da to učini.
2. **Otklanjanje grešaka i testiranje**: Prilikom razvoja i testiranja aplikacija vođenih AI-jem, mogućnost prisilnog pozivanja funkcija je neprocjenjiva za potrebe otklanjanja grešaka. Eksplicitnim pokretanjem određenih funkcija možete izolirati i testirati pojedinačne komponente vaše aplikacije. To vam omogućava da provjerite ispravnost implementacija funkcija, validirate ulazne parametre i osigurate da se vraćaju očekivani rezultati.
3. **Rukovanje graničnim slučajevima**: Mogu postojati granični slučajevi ili izuzetni scenariji gdje proces dinamičkog odabira AI-ja možda neće odabrati izvršavanje funkcije koju bi trebao, a vi to znate na osnovu vanjskih procesa. U takvim slučajevima, mogućnost prisilnog pozivanja funkcije vam omogućava da eksplicitno rukujete ovim situacijama. Definirajte pravila ili uslove u logici vaše aplikacije da odredite kada treba zaobići diskreciju AI-ja.

4. **Konzistentnost i ponovljivost**: Ako imate određeni slijed funkcija koje se trebaju izvršiti određenim redoslijedom, prisilno pozivanje garantuje da će se isti slijed pratiti svaki put. Ovo je posebno važno u aplikacijama gdje su konzistentnost i predvidljivo ponašanje kritični, kao što su finansijski sistemi ili naučne 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 direktno pozvati potrebnu funkciju. Ovo može pomoći u smanjenju latencije i poboljšanju ukupne efikasnosti vaše aplikacije.

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

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

Parametar odabira alata

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

- `auto` daje AI-ju punu diskreciju nad korištenjem alata ili jednostavnim odgovaranjem

- `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 od onih koje su mu dostupne, čak i ako nijedna zaista ne odgovara upitu. U vrijeme objavljivanja, nije mi poznato da postoji model koji će vratiti prazan `tool_calls` odgovor ili na neki drugi način dati do znanja da nije pronašao odgovarajuću funkciju za poziv.

Prisilno Pozivanje Funkcije za Dobivanje Strukturiranog Izlaza

Mogućnost prisilnog pozivanja funkcije vam daje način da dobijete strukturirane podatke iz chat završetka umjesto da ih sami morate izvlačiti iz običnog tekstualnog odgovora.

> Zašto je prisilno pozivanje funkcija za dobivanje strukturiranog izlaza toliko važno? Jednostavno rečeno, zato što je izvlačenje strukturiranih podataka iz izlaza VJM-a prava noćna mora. Možete si malo olakšati život tražeći podatke u XML-u, ali onda morate parsirati XML. I šta učiniti kada taj XML nedostaje jer je vaša UI odgovorila: "Žao mi je, ali ne mogu generirati podatke koje ste zatražili jer bla, bla, bla..."

Kada koristite alate na ovaj način:

- Vjerovatno biste trebali definirati jedan alat u svom zahtjevu

- Ne zaboravite prisiliti korištenje njegove funkcije pomoću parametra `tool_choice`.
- Zapamtite da će model proslijediti unos alatu, tako da naziv alata i opis trebaju biti iz perspektive modela, ne vaše.

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

Evo primjera korištenja Claude 3 za bilježenje sažetka slike u dobro strukturirani JSON, ovaj put iz komandne linije koristeći `curl`:

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

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

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

U datom primjeru, koristimo Claude 3 model od Anthropic-a za generisanje strukturiranog JSON sažetka slike. Evo kako to funkcioniše:

1. Definišemo jedan alat pod nazivom `record_summary` u nizu `tools` korisnog tereta zahtjeva. Ovaj alat je zadužen za bilježenje sažetka slike u dobro strukturirani JSON.
2. Alat `record_summary` ima `input_schema` koja određuje očekivanu strukturu JSON izlaza. Definiše tri svojstva:
    - `key_colors`: Niz 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`: Svojstvo tipa string za kratak opis slike, ograničeno na 1-2 rečenice.
    - `estimated_year`: Opcionalno svojstvo tipa integer za procijenjenu godinu kada je slika nastala, ako se čini da je u pitanju stvarna fotografija.
3. U nizu `messages`, dostavljamo podatke slike kao base64-kodirani string zajedno sa tipom medija. Ovo omogućava modelu da obradi sliku kao dio ulaza.
4. Također dajemo uputstvo Claude-u da koristi alat `record_summary` za opisivanje slike.

5. Kada se zahtjev pošalje Claude 3 modelu, on analizira sliku i generiše JSON sažetak na osnovu specificirane `input_schema`. Model izdvaja ključne boje, daje kratak opis i procjenjuje godinu nastanka slike (ako je primjenjivo).
6. Generisani JSON sažetak se prosljeđuje kao parametri alatu `record_summary`, pružajući strukturiranu reprezentaciju ključnih karakteristika slike.

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

Mogućnost da se forsira poziv funkcije i specificira očekivana struktura izlaza je moćna karakteristika upotrebe alata u aplikacijama vođenim vještačkom inteligencijom. To omogućava programerima da imaju više kontrole nad generisanim izlazom i pojednostavljuje integraciju podataka generisanih od strane AI-a u tok rada njihove aplikacije.

Izvršavanje Funkcije(a)

Definisali ste funkcije i dali upute vašem AI-u, 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 njene parametre odgovarajućoj implementaciji *u vašem aplikacijskom kodu.*

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

Ponekad je poziv funkcije kraj operacije, ali ako rezultati predstavljaju informacije u lancu razmišljanja koji AI treba nastaviti, tada vaš aplikacijski kod treba ubaciti rezultate izvršenja u transkript razgovora i pustiti AI da nastavi sa obradom.

Na primjer, evo Raix deklaracije funkcije koju koristi Olympia-in `AccountManager` za komunikaciju sa našim klijentima kao dio Inteligentne Orkestracije Toka Rada za korisničku podršku.

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

  # lots of other functions...

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

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

1. Klasa `AccountManager` definiše mnoge funkcije vezane za upravljanje računom. Može mijenjati vaš plan, dodavati i uklanjati članove tima, između ostalog.
2. Njegove instrukcije najvišeg nivoa govore `AccountManager`-u da treba obavijestiti vlasnika računa o rezultatima zahtjeva za promjenu računa, koristeći funkciju `notify_account_owner`.
3. Koncizna definicija funkcije uključuje njene:

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

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

Kada AI model primi novi zahtjev za završetak razgovora s ažuriranim transkriptom, ima pristup rezultatima prethodno izvršene funkcije. Može analizirati ove rezultate, uključiti ih u svoj proces odlučivanja i generisati sljedeći odgovor ili akciju na osnovu kumulativnog konteksta razgovora. Može odabrati da izvrši dodatne funkcije na osnovu ažuriranog konteksta, ili može generisati konačni odgovor na originalni upit ako utvrdi da dodatni pozivi funkcija nisu potrebni.

Opcionalni nastavak originalnog upita

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

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

Zavisno od modela koji se koristi, rezultati poziva funkcije će živjeti u porukama transkripta koje imaju svoju posebnu ulogu ili će biti prikazani u nekoj drugoj sintaksi. Ali važan dio je da ti podaci budu u transkriptu, tako da ih AI može razmotriti dok odlučuje šta dalje da radi.

Česta (i potencijalno skupa) greška je zaboraviti dodati rezultate funkcije u transkript prije nastavka razgovora. Kao rezultat, AI će dobiti upit na praktično isti način kao i prije nego što je prvi put pozvao funkciju. Drugim riječima, što se tiče AI-ja, on još nije pozvao funkciju. Tako da je pozove ponovo. I ponovo. I ponovo, 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

Obezbijedite jasne i opisne nazive i opise za svaki alat i njegove ulazne parametre. Ovo 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 imena funkcija ili formulacije opisa. Ponekad uklanjanje opisa *poboljšava* performanse.

Obrada rezultata alata

Kada proslijeđujete rezultate alata nazad VJM-u, osigurajte da su dobro strukturirani i sveobuhvatni. Koristite smislene ključeve i vrijednosti za predstavljanje izlaza svakog alata. Eksperimentišite s različitim formatima i vidite koji najbolje radi, od JSON-a do običnog teksta.

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

Upravljanje greškama

Implementirajte robusne mehanizme za upravljanje greškama kako biste obradili slučajeve gdje VJM može generisati nevažeće ili nepodržane ulazne parametre za pozive alata.

Elegantno rukujte i oporavite se od bilo kakvih grešaka koje se mogu pojaviti tokom izvršavanja alata.

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

Na primjer, evo pojednostavljene verzije implementacije Google pretrage 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 proces koji se odvija u dva koraka. Prvo se izvrši pretraživanje, zatim se sumiraju rezultati. Ako dođe do greške, bez obzira kakva ona bila, poruka o izuzetku se pakuje i šalje nazad AI-ju. Ova tehnika je temelj praktično svih obrazaca *Inteligentnog rukovanja greškama.*

Na primjer, recimo da `GoogleSearch` API poziv ne uspije zbog 503 greške da servis nije dostupan. To se propagira do rescue bloka najvišeg nivoa, i opis greške se šalje nazad AI-ju kao rezultat izvršavanja funkcije. Umjesto da korisniku prikaže prazan ekran ili tehničku grešku, AI kaže nešto poput "Žao mi je, ali trenutno ne mogu pristupiti svojim mogućnostima Google pretraživanja. Mogu pokušati kasnije, ako želite."

Ovo možda izgleda kao pametan trik, ali razmotrite drugačiju vrstu greške, onu gdje AI poziva vanjski API i ima direktnu kontrolu nad parametrima koji se prosljeđuju API-ju. Možda je napravio grešku u načinu na koji je generisao te parametre? Pod uslovom da je poruka o grešci iz vanjskog API-ja dovoljno detaljna, proslijeđivanje poruke o grešci

nazad AI-ju koji je izvršio poziv znači da on može preispitati te parametre i pokušati ponovo. Automatski. Bez obzira kakva je greška bila.

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

Iterativno poboljšanje

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

1. Počnite sa jednostavnim definicijama alata: Počnite definisanjem alata sa jasnim i konciznim imenima, opisima i ulaznim parametrima. U početku izbjegavajte prekomplikovano postavljanje alata i fokusirajte se na osnovnu funkcionalnost. Na primjer, ako želite sačuvati rezultate analize sentimenta, počnite sa osnovnom definicijom poput:

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

2. Testirajte i posmatrajte: Nakon što postavite početne definicije alata, testirajte ih različitim upitima i posmatrajte kako VJM komunicira sa alatom. Obratite pažnju

na kvalitet i relevantnost generisanih odgovora. Ako VJM generiše suboptimalne odgovore, vrijeme je da se definicije alata usavrše.

3. Usavršite opise: Ako VJM pogrešno tumači svrhu alata, pokušajte usavršiti opis alata. Pružite više konteksta, primjera ili pojašnjenja kako biste usmjerili VJM ka efektivnom korištenju 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, razmotrite prilagođavanje definicija parametara. Dodajte specifičnija ograničenja, pravila validacije ili primjere kako biste pojasnili očekivani format unosa.
5. Iteriranje na osnovu povratnih informacija: Kontinuirano pratite performanse vaših alata i prikupljajte povratne informacije od korisnika i zainteresiranih strana. Koristite ove povratne informacije za identifikaciju područja za poboljšanje 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 poboljšanjem definicija alata na osnovu posmatranog ponašanja i povratnih informacija, možete postepeno poboljšati performanse i efikasnost vaše aplikacije zasnovane na vještačkoj inteligenciji. Zapamtite da definicije alata trebaju biti jasne, sažete i fokusirane na specifični zadatak. Redovno testirajte i potvrđujte interakcije alata kako biste osigurali da su u skladu sa željenim ishodima.

Komponovanje i ulančavanje alata

Jedan od najmoćnijih aspekata upotrebe alata koji je do sada samo nagovješten je mogućnost komponovanja i ulančavanja više alata zajedno za izvršavanje složenih zadataka. Pažljivim dizajniranjem definicija alata i njihovih ulaznih/izlaznih formata, možete kreirati ponovno upotrebljive građevinske blokove koji se mogu kombinovati na različite načine.

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

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

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

1. LLM prima upit korisnika koji traži uvid u podatke o prodaji za određenu kategoriju proizvoda.
2. LLM odabire alat `DataRetrieval` i generiše odgovarajuće ulazne parametre za preuzimanje relevantnih podataka o prodaji iz baze podataka.
3. Preuzeti podaci se "proslijeđuju" alatu `DataProcessing`, koji izračunava metrike kao što su ukupni prihod, prosječna prodajna cijena i stopa rasta.
4. Obrađeni podaci se zatim procesiraju kroz alat `DataVisualization`, koji kreira vizuelno privlačan grafikon ili dijagram za predstavljanje uvida, vraćajući URL grafikona nazad LLM-u.
5. Na kraju, LLM generiše formatiran odgovor na upit korisnika koristeći markdown, uključujući vizualizirane podatke i pružajući sažetak ključnih nalaza.

Komponovanjem ovih alata zajedno, možete kreirati besprijekoran tok rada za analizu podataka koji se može lako integrisati u vašu aplikaciju. Ljepota ovog pristupa je u tome što se svaki alat može razvijati i testirati nezavisno, a zatim kombinovati na različite načine za rješavanje različitih problema.

Da bi se omogućilo glatko komponovanje i ulančavanje alata, važno je definisati jasne ulazne i izlazne formate za svaki alat.

Na primjer, alat `DataRetrieval` može prihvatiti parametre kao što su detalji povezivanja s bazom podataka, ime tabele i uslovi upita, te vratiti skup rezultata kao strukturirani JSON objekt. Alat `DataProcessing` zatim može očekivati ovaj JSON objekt kao ulaz i proizvesti transformirani JSON objekt kao izlaz. Standardizacijom protoka podataka između alata možete osigurati kompatibilnost i mogućnost ponovne upotrebe.

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

Zapamtite, snaga upotrebe alata leži u fleksibilnosti i modularnosti koju pruža. Razbijanjem složenih zadataka na manje, ponovno upotrebljive alate, možete kreirati robusnu i prilagodljivu aplikaciju zasnovanu na vještačkoj inteligenciji koja može riješiti širok spektar izazova.

Budući pravci

Kako se oblast razvoja aplikacija zasnovanih na vještačkoj inteligenciji razvija, možemo očekivati dalje napretke u mogućnostima upotrebe alata. Neki potencijalni budući pravci uključuju:

1. **Višestruka upotreba alata**: LLM-ovi bi mogli biti u stanju odlučiti koliko puta trebaju koristiti alate kako bi generisali zadovoljavajući odgovor. To bi moglo uključivati više rundi odabira i izvršavanja alata na osnovu međurezultata.
2. **Predefinisani alati**: AI platforme bi mogle pružiti set predefinisanih alata koje programeri mogu koristiti odmah, kao što su Python interpretatori, alati za pretraživanje weba ili uobičajene utility funkcije.
3. **Besprijekorna integracija**: Kako upotreba alata postaje sve zastupljenija, možemo očekivati bolju integraciju između AI platformi i popularnih razvojnih okvira, olakšavajući programerima da uključe upotrebu alata u svoje aplikacije.

Upotreba alata je moćna tehnika koja omogućava programerima da iskoriste puni potencijal LLM-ova u aplikacijama zasnovanim na vještačkoj inteligenciji. Povezivanjem

LLM-ova s vanjskim alatima i resursima, možete kreirati dinamičnije, inteligentnije i kontekstualno svjesnije sisteme koji se mogu prilagoditi potrebama korisnika i pružiti vrijedne uvide i akcije.

Iako upotreba 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 sistema. Morate se nositi sa scenarijima gdje pozivi alata mogu biti neuspješni, vratiti neočekivane rezultate ili imati implikacije na performanse. Dodatno, trebali biste razmotriti sigurnosne mjere i kontrolu pristupa kako biste spriječili neovlaštenu ili zlonamjernu upotrebu alata. Pravilno rukovanje greškama, bilježenje i mehanizmi praćenja su ključni za održavanje integriteta i performansi vaše aplikacije zasnovane na vještačkoj inteligenciji.

Dok istražujete mogućnosti upotrebe alata u vašim vlastitim projektima, zapamtite da trebate početi s jasnim ciljevima, dizajnirati dobro strukturirane definicije alata i iterirati na osnovu povratnih informacija i rezultata. Uz pravi pristup i način razmišljanja, upotreba alata može otključati nove nivoe inovacija i vrijednosti u vašim aplikacijama vođenim vještačkom inteligencijom

Obrada toka podataka

Prijenos podataka preko HTTP-a, također poznat kao eventi poslani sa servera (SSE), je mehanizam gdje server kontinuirano šalje podatke klijentu kako postaju dostupni, bez potrebe da ih klijent eksplicitno zatraži. Budući da se AI-jev odgovor generira postepeno, ima smisla pružiti responzivno korisničko iskustvo prikazivanjem AI-jevog izlaza dok se generira. I zapravo, svi API-ji AI provajdera koje poznajem nude opciju prijenosa podataka u svojim krajnjim tačkama za završavanje.

Razlog zašto se ovo poglavlje pojavljuje ovdje u knjizi, odmah nakon Korištenje alata je zbog toga koliko moćno može biti kombinovanje korištenja alata sa AI odgovorima uživo prema korisnicima. To omogućava dinamična i interaktivna iskustva gdje AI može obraditi korisnički unos, koristiti različite alate i funkcije po vlastitom nahođenju, te pružati odgovore u realnom vremenu.

Da bi se postigla ova besprijekorna interakcija, potrebno je napisati rukovaoce toka koji mogu otpremati pozive alatnih funkcija koje poziva AI, kao i izlaz običnog teksta prema

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

Implementacija ReplyStream-a

Da bismo pokazali kako se obrada toka može implementirati, ovo poglavlje će detaljno proučiti pojednostavljenu verziju klase `ReplyStream` koja se koristi u Olympia-i. 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 Olympia-inom `PromptSubscriber`-u, koji osluškuje putem Wisper-a kreiranje 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...
```

Pored `context` reference na pretplatnika promptova koji ga je instancirao, klasa `ReplyStream` također ima instance varijable za pohranjivanje bafera primljenih podataka, te nizove za praćenje imena funkcija i argumenata koji se pozivaju tokom procesiranja streama.

```
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`, inicijalizira bafer, kontekst i druge varijable.

Metoda `call` je glavna ulazna tačka za obradu streaming podataka. Prima `chunk` podataka (predstavljen kao heš) i opcionalni parametar `bytesize`, koji u našem primjeru nije korišten. Unutar ove metode, klasa koristi podudaranje uzoraka za rukovanje različitim scenarijima na osnovu strukture primljenog dijela podataka.

Pozivanje `deep_symbolize_keys` na chunk čini podudaranje uzoraka elegantnijim, omogućavajući nam da radimo 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 njegovim nazivom funkcije. Ako ga otkrijemo, smještamo ga u niz `f_name`. Nazive funkcija pohranjujemo u indeksirani niz, jer model može vršiti paralelno pozivanje funkcija, šaljući više od jedne funkcije na izvršavanje istovremeno.

> Paralelno pozivanje funkcija je sposobnost AI modela da izvršava više poziva funkcija zajedno, omogućavajući da se efekti i rezultati ovih poziva funkcija riješe paralelno. Ovo je posebno korisno ako funkcije zahtijevaju duže vrijeme izvršavanja, i smanjuje broj povratnih putovanja sa API-jem, što zauzvrat može uštedjeti značajnu količinu potrošnje 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 nazivima funkcija, argumente spremamo u indeksirani niz.

Zatim, tražimo obične poruke namijenjene korisniku, koje će stizati sa servera jedan token po jedan i biti dodijeljene varijabli `new_content`. Također moramo pratiti `finish_reason`. On će biti `nil` sve do posljednjeg dijela 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 da dodamo izraz za podudaranje obrazaca kako bismo obradili poruke o greškama koje šalje pružalac AI modela. U lokalnim razvojnim okruženjima, podižemo izuzetak, ali u produkciji, grešku 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 će se izvršiti ako se nijedan od prethodnih obrazaca nije poklopio. To je samo sigurnosna mjera kako bismo saznali ako AI model počne slati neprepoznate dijelove.

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

Metoda `send_to_client` je odgovorna za slanje međuspremljenog sadržaja klijentu. Ona provjerava da međuspremnik nije prazan, ažurira sadržaj poruke bota, prikazuje poruku bota i 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` se poziva kada je obrada toka završena. Ona izvršava pozive funkcija ako su neki primljeni tokom toka, ažurira poruku bota sa konačnim sadržajem i drugim relevantnim informacijama, te resetuje historiju 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, potrebno je "dispečirati" taj poziv funkcije (naziv i argumente) na takav način da se izvrši i da se poruke `function_call` i `function_result` dodaju u transkript razgovora

Prema mom iskustvu, bolje je upravljati kreiranjem poruka funkcija na jednom mjestu u vašoj bazi koda, umjesto da se oslanjate na implementacije alata. To nije samo čišće rješenje, 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 ponovo.* Potencijalno beskonačno. Zapamtite da je AI potpuno bez stanja, pa ako ne vratite te pozive funkcija nazad u model, za njega 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 historije poziva funkcija nakon izvršavanja je jednako važno kao i osiguravanje da se poziv i rezultati pojave u vašem transkriptu, kako ne biste stalno iznova pozivali iste funkcije svaki put kada se petlja izvršava.

“Konverzacijska petlja”

Ja stalno spominjem petlje, ali ako ste novi u pozivanju funkcija, možda nije očigledno *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, a zatim ponovno pošaljete originalni prompt kako biste dobili novi set poziva funkcija ili rezultate namijenjene korisniku.

U klasi `PromptSubscriber`, koristimo metodu `prompt` iz modula `PromptDeclarations` da definiramo ponašanje konverzacijske petlje. Parametar `until` je postavljen 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 se `bot_message` označava kao završen? Ako ste zaboravili, pogledajte ponovo liniju 13 metode `finalize`.

Pregledajmo kompletnu logiku obrade toka.

1. `PromptSubscriber` prima novu korisničku poruku putem metode `message_created`, koju poziva Wisper sistem za pretplatu i objavu svaki put kada krajnji korisnik kreira novi prompt.
2. Metoda klase `prompt` deklarativno definiše ponašanje logike završetka razgovora za `PromptSubscriber`. AI model će izvršiti završetak razgovora sa sadržajem korisničke poruke, novom instancom `ReplyStream` kao parametrom toka, i određenim uslovom petlje.
3. AI model obrađuje prompt i počinje generisati odgovor. Kako se odgovor prenosi tokom, metoda `call` instance `ReplyStream` se poziva za svaki dio podataka.
4. Ako AI model odluči pozvati funkciju alata, naziv funkcije i argumenti se izdvajaju iz dijela 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. Kada je obrada toka završena, poziva se metoda `finalize`. Ako su tokom toka pozvane bilo koje funkcije alata, one se izvršavaju koristeći metodu `dispatch` klase `PromptSubscriber`.
7. Metoda `dispatch` dodaje poruku `function_call` u transkript razgovora, izvršava odgovarajuću funkciju alata i dodaje poruku `function_result` u transkript sa rezultatom poziva funkcije.
8. Nakon izvršavanja funkcija alata, historija poziva funkcija se briše kako bi se spriječili dupli pozivi funkcija u narednim petljama.

9. Ako nije pozvana nijedna funkcija alata, metoda `finalize` ažurira `bot_message` sa konačnim sadržajem, označava ga kao završen i šalje ažuriranu poruku klijentu.
10. Evaluira se uslov petlje `-> { bot_message.complete? }`. Ako `bot_message` nije označen kao završen, petlja se nastavlja, i originalni prompt se ponovo šalje sa ažuriranim transkriptom razgovora.
11. Koraci 3-10 se ponavljaju sve dok `bot_message` ne bude označen kao završen, što ukazuje da je AI model završio generiranje svog odgovora i da nema potrebe za izvršavanjem dodatnih funkcija alata.

Implementacijom ove petlje razgovora, omogućavate AI modelu da se upusti u dvosmjernu interakciju sa aplikacijom, izvršavajući funkcije alata po potrebi i generirajući odgovore vidljive korisniku sve dok razgovor ne dođe do prirodnog zaključka.

Kombinacija obrade toka i petlje razgovora omogućava dinamična i interaktivna AI iskustva, gdje AI model može obrađivati korisnički unos, koristiti različite alate i funkcije, te pružati odgovore u realnom vremenu na osnovu konteksta razgovora koji se razvija.

Automatski Nastavak

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

U odgovoru toka sa AI platforme, ovu situaciju možete otkriti pregledanjem varijable `finish_reason` u dijelu. Ako je `finish_reason` postavljen na `"length"` (ili neku drugu ključnu vrijednost specifičnu za model), to znači da je model dostigao svoje maksimalno ograničenje tokena tokom generiranja i da je izlaz prekinut.

Jedan od načina da se ova situacija elegantno riješi i pruži besprijekorno korisničko iskustvo je implementacija mehanizma automatskog nastavka u vašoj logici obrade

toka. Dodavanjem obrasca podudaranja za razloge završetka vezane za dužinu, možete odabrati da se petlja nastavi i automatski nastavi izlaz od mjesta gdje je stao.

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 finaliziranja toka, dodajemo par poruka u transkript bez finaliziranja, premještamo orig-

inalnu poruku vidljivu korisniku na "dno" transkripta ažuriranjem njenog `created_at` atributa, a zatim dopuštamo da se petlja nastavi, tako da AI nastavlja generisanje tamo gdje je stao.

Zapamtite da je krajnja tačka za AI završavanje bez stanja. Ona "zna" samo ono što joj kažete putem transkripta. U ovom slučaju, način na koji AI-u komuniciramo da je bio prekinut je dodavanjem "nevidljivih" (za krajnjeg korisnika) poruka u transkript. Međutim, zapamtite da je ovo namjerno pojednostavljeni primjer. Stvarna implementacija bi trebala izvršiti dodatno upravljanje transkriptom kako bi se osiguralo da ne trošimo tokene i/ili ne zbunjujemo AI dupliciranim porukama asistenta u transkriptu.

Stvarna implementacija automatskog nastavljanja bi također trebala imati takozvanu "logiku prekidača" kako bi se spriječilo nekontrolisano petljanje. Razlog tome je što bi, s obzirom na određene vrste korisničkih upita i niske postavke `max_tokens`, AI mogao beskonačno nastaviti petljati izlaz vidljiv korisniku.

> Imajte na umu da svaka petlja zahtijeva zaseban zahtjev, i da svaki zahtjev ponovo konzumira cijeli vaš transkript. Definitivno 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, naročito kada se koriste premium komercijalni modeli.

Zaključak

Obrada toka podataka je ključni aspekt izgradnje aplikacija pogonjenih AI-em koje kombinuju upotrebu alata sa AI odgovorima uživo. Efikasnim rukovanjem tokova podataka iz API-ja AI platformi, možete pružiti besprijekorno i interaktivno korisničko iskustvo, upravljati velikim odgovorima, optimizirati korištenje resursa i elegantno rukovati greškama.

Predstavljena klasa `Conversation::ReplyStream` demonstrira kako obrada toka može biti implementirana u Ruby aplikaciji koristeći podudaranje uzoraka i arhitekturu vođenu događajima. Razumijevanjem i korištenjem tehnika obrade toka, možete otključati puni potencijal AI integracije u vašim aplikacijama i isporučiti moćna i privlačna korisnička iskustva.

Samozacjeljujući podaci

Samozacjeljujući podaci predstavljaju moćan pristup osiguravanju integriteta, konzistentnosti i kvaliteta podataka u aplikacijama koristeći mogućnosti velikih jezičkih modela (LLM). Ova kategorija obrazaca fokusira se na ideju korištenja vještačke inteligencije za automatsko otkrivanje, dijagnosticiranje i ispravljanje anomalija, nekonzistentnosti ili grešaka u podacima, čime se smanjuje opterećenje programera i održava visok nivo pouzdanosti podataka.

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

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

interpretiraju obrasce, odnose i anomalije unutar podataka. Koristeći svoje sposobnosti obrade i razumijevanja prirodnog jezika, mogu identificirati potencijalne probleme ili nekonzistentnosti u podacima i poduzeti odgovarajuće akcije za njihovo ispravljanje.

Proces samozacjeljivanja podataka obično uključuje nekoliko ključnih koraka:

1. **Praćenje podataka**: AI radnici konstantno prate tokove podataka aplikacije, baze podataka ili sisteme za pohranu, tražeći bilo kakve znakove anomalija, nekonzistentnosti ili grešaka. Alternativno, možete aktivirati AI komponentu kao reakciju na izuzetak.
2. **Detekcija anomalija**: Kada se otkrije problem, AI radnik detaljno analizira podatke kako bi identificirao specifičnu prirodu i opseg problema. To može uključivati otkrivanje nedostajućih vrijednosti, nekonzistentnih formata ili podataka koji krše predefinirana pravila ili ograničenja.
3. **Dijagnoza i korekcija**: Nakon što se problem identificira, AI radnik koristi svoje znanje i razumijevanje domene podataka da odredi odgovarajući tok akcije. To može uključivati automatsko ispravljanje podataka, popunjavanje nedostajućih vrijednosti ili označavanje problema za ljudsku intervenciju ako je potrebno.
4. **Kontinuirano učenje (opcionalno, zavisno od slučaja upotrebe)**: Kako vaš AI radnik susreće i rješava različite probleme s podacima, može proizvesti opis onoga što se dogodilo i kako je reagovao. Ovi metapodaci mogu se unijeti u procese učenja koji vam omogućavaju (i možda osnovnom modelu, putem finog podešavanja) da postanete efikasniji tokom vremena u identificiranju i rješavanju anomalija podataka.

Automatskim otkrivanjem i ispravljanjem problema s podacima, možete osigurati da vaša aplikacija radi s visokokvalitetnim, pouzdanim podacima. To smanjuje rizik od grešaka, nekonzistentnosti ili bugova vezanih za podatke koji bi mogli utjecati na funkcionalnost aplikacije ili korisničko iskustvo.

Jednom kada imate AI radnike koji se bave zadatkom praćenja i ispravljanja podataka, možete usmjeriti svoje napore na druge kritične aspekte aplikacije. Ovo štedi vrijeme

i resurse koji bi inače bili potrošeni na ručno čišćenje i održavanje podataka. Zapravo, kako vaše aplikacije rastu u veličini i složenosti, ručno upravljanje kvalitetom podataka postaje sve izazovnije. Obrasci "Samozacjeljujućih podataka" efikasno se skaliraju koristeći moć vještačke inteligencije za rukovanje velikim količinama podataka i otkrivanje problema u realnom vremenu.

Zbog svoje prirode, AI modeli se mogu prilagoditi promjenljivim obrascima podataka, shemama ili zahtjevima tokom vremena uz malo ili nimalo nadzora. Sve dok njihove direktive pružaju adekvatne smjernice, posebno u pogledu željenih rezultata, vaša aplikacija može evoluirati i rukovati 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 razgovarali, kao što je "Mnoštvo radnika". Mogućnost samozacjeljivanja podataka može se posmatrati kao specijalizirana vrsta radnika koji se fokusira specifično na osiguravanje kvaliteta i integriteta podataka. Ova vrsta radnika djeluje zajedno s drugim AI radnicima, pri čemu svaki doprinosi različitim aspektima funkcionalnosti aplikacije.

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

Praktična studija slučaja: Popravljanje neispravnog JSON-a

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

Ova tehnika se može primijeniti na uobičajeni izazov rukovanja nesavršenim ili nekonzistentnim podacima koje generiraju LLM-ovi, kao što je neispravan JSON, i pruža pristup za automatsko otkrivanje i ispravljanje ovih problema.

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

Da bih riješio ovaj problem, implementirao sam praktično rješenje u obliku JsonFixer klase. Ova klasa utjelovljuje obrazac "Samozacjeljujućih podataka" tako što uzima neispravan JSON kao ulaz i koristi LLM da ga popravi, pri tome čuvajući što je moguće više informacija i prvobitne namjere.

```
class JsonFixer
  include Raix::ChatCompletion

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

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

    self.stop = ["```"]

    chat_completion(json: true)
  end

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

```
23    end
24  end
```

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

Proces samopopravljanja JSON podataka funkcioniše na sljedeći način:

1. **Generisanje JSON-a**: LLM se koristi za generisanje JSON podataka na osnovu određenih uputa ili zahtjeva. Međutim, zbog prirode LLM-ova, generirani JSON ne mora uvijek biti savršeno validan. JSON parser će naravno podići `ParserError` ako mu date nevažeći JSON.

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

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

2. **Korekcija zasnovana na LLM-u**: Klasa `JSONFixer` šalje neispravan JSON nazad LLM-u, zajedno sa specifičnim promptom ili instrukcijom da popravi JSON uz maksimalno očuvanje originalnih informacija i namjere. LLM, treniran na ogromnim količinama podataka i s razumijevanjem JSON sintakse, pokušava ispraviti greške i generirati validan JSON string. Ograđivanje odgovora se koristi za ograničavanje izlaza LLM-a, a mi biramo 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 nazad u tok rada aplikacije, omogućavajući aplikaciji da nastavi obrađivati podatke bez prekida. Loš JSON je "izliječen".

Iako sam napisao i prepisao vlastitu `JSONFixer` implementaciju 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 obrasca samozacjeljujućih podataka. Proces korekcije zasnovan na LLM-u ima za cilj očuvati originalne informacije i namjeru generiranog JSON-a što je više moguće. Ovo osigurava da popravljeni JSON zadržava svoje semantičko značenje i može se efektivno 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 izazova s podacima. Pokazuje snagu kombinovanja tradicionalnih programerskih tehnika sa AI mogućnostima za izgradnju robusnih i efikasnih aplikacija.

Postelov zakon i obrazac "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 kaže:

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

Ovaj princip, koji je izvorno artikulirao Jon Postel, pionir ranog Interneta, naglašava

važnost izgradnje sistema koji su tolerantni prema raznovrsnim ili čak blago netačnim ulazima, dok istovremeno održavaju strogo pridržavanje specificiranih protokola pri slanju izlaza.

U kontekstu "Samozacjeljujućih podataka", klasa JSONFixer utjelovljuje Postelov zakon tako što je liberalna u prihvatanju pokvarenih 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 prihvatanju nesavršenog JSON-a, klasa JSONFixer pokazuje robusnost i fleksibilnost. Priznaje da podaci u stvarnom svijetu često dolaze u različitim oblicima i možda se neće uvijek pridržavati strogih specifikacija. Gracioznim rukovanjem i ispravljanjem ovih odstupanja, klasa osigurava da aplikacija može nastaviti glatko funkcionirati, čak i u prisustvu 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 tačnost podataka prije nego što ih proslijedi drugim dijelovima aplikacije. Ovaj konzervativni pristup garantuje da je izlaz klase JSONFixer pouzdan i konzistentan, promovirajući interoperabilnost i sprečavajući širenje grešaka.

Zanimljive činjenice o Jonu Postelu:

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

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

Klasa `JSONFixer` promoviše robusnost, fleksibilnost i interoperabilnost, što su bile osnovne vrijednosti koje je Postel zastupao tokom svoje karijere. Izgradnjom sistema koji su tolerantni na nesavršenosti, dok istovremeno održavaju strogo pridržavanje protokola, možemo kreirati aplikacije koje su otpornije i prilagodljivije stvarnim izazovima.

Razmatranja i kontraindikacije

Primjenjivost pristupa samozacjeljujućih podataka u potpunosti zavisi od vrste podataka kojima vaša aplikacija upravlja. Postoji razlog zašto možda ne želite jednostavno monkeypatch-ovati `JSON.parse` da automatski ispravlja *sve JSON greške parsiranja* u vašoj aplikaciji: ne mogu i ne trebaju se sve greške automatski ispravljati.

Samozacjeljivanje je posebno problematično kada je povezano s regulatornim zahtjevima ili zahtjevima usklađenosti vezanim za rukovanje i obradu podataka. Neke industrije, poput zdravstva i finansija, imaju tako stroge propise u vezi s integritetom podataka i mogućnošću revizije da bilo kakva "black box" korekcija podataka bez odgovarajućeg nadzora ili evidencije može prekršiti ove propise. Ključno je osigurati da sve tehnike samozacjeljivanja podataka koje osmislite budu usklađene s primjenjivim pravnim i regulatornim okvirima.

Primjena tehnika samozacjeljujućih podataka, posebno onih koji uključuju AI modele, također može imati veliki uticaj na performanse aplikacije i korištenje resursa. Obrada velikih količina podataka kroz AI modele za otkrivanje i ispravljanje grešaka može biti računski zahtjevna. Važno je procijeniti kompromise između prednosti samozacjeljujućih podataka i povezanih troškova performansi i resursa.

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

Kritičnost podataka

Prilikom razmatranja primjene tehnika samozacjeljujućih podataka, ključno je procijeniti kritičnost podataka koji se obrađuju. Nivo kritičnosti odnosi se na važnost i osjetljivost podataka u kontekstu vaše aplikacije i njenog poslovnog domena.

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

1. **Finansijske transakcije**: U finansijskim aplikacijama, kao što su bankarski sistemi ili trgovačke platforme, tačnost podataka je od najveće važnosti. Čak i manje greške u finansijskim podacima mogu imati značajne posljedice, kao što su netačna stanja računa, pogrešno usmjerena sredstva ili pogrešne odluke o trgovanju. U ovim slučajevima, automatske korekcije bez temeljite verifikacije i revizije mogu uvesti neprihvatljive rizike.
2. **Medicinski zapisi**: Zdravstvene aplikacije se bave vrlo osjetljivim i povjerljivim podacima pacijenata. Netačnosti u medicinskim zapisima mogu imati ozbiljne implikacije za sigurnost pacijenata i odluke o liječenju. Automatsko modificiranje medicinskih podataka bez odgovarajućeg nadzora i validacije od strane kvalificiranih zdravstvenih radnika može prekršiti regulatorne zahtjeve i ugroziti dobrobit pacijenata.

3. **Pravni dokumenti**: Aplikacije koje rukuju pravnim dokumentima, kao što su ugovori, sporazumi ili sudski podnesci, zahtijevaju strogu tačnost i integritet. Čak i manje greške u pravnim podacima mogu imati značajne pravne posljedice. Automatske korekcije u ovom domenu možda nisu prikladne, jer podaci često zahtijevaju ručni pregled i verifikaciju od strane pravnih stručnjaka kako bi se osigurala njihova validnost i izvršnost.

U ovim kritičnim scenarijima podataka, rizici povezani s automatskim korekcijama često nadmašuju potencijalne koristi. Posljedice uvođenja grešaka ili netačnog modificiranja podataka mogu biti ozbiljne, dovodeći do finansijskih gubitaka, pravnih odgovornosti, pa čak i štete pojedincima.

Kada se radi s vrlo kritičnim podacima, neophodno je dati prioritet procesima ručne verifikacije i validacije. Ljudski nadzor i stručnost su ključni u osiguravanju tačnosti i integriteta podataka. Automatizirane tehnike samozacjeljivanja se i dalje mogu koristiti za označavanje potencijalnih grešaka ili nedosljednosti, ali konačna odluka o korekcijama treba uključivati ljudsku procjenu i odobrenje.

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

Ključ je pažljivo procijeniti kritičnost svake kategorije podataka u vašoj aplikaciji i definirati jasne smjernice i procese za rukovanje korekcijama na osnovu 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.

Na kraju, odluka o primjeni tehnika samozacjeljivanja podataka na kritične podatke treba biti donesena u konsultaciji sa stručnjacima iz domene, pravnim savjetnicima

i drugim relevantnim zainteresiranim stranama. Neophodno je razmotriti specifične zahtjeve, propise i rizike povezane s podacima vaše aplikacije i uskladiti strategije korekcije podataka u skladu s tim.

Ozbiljnost greške

Prilikom primjene tehnika samozacjeljivanja podataka, važno je procijeniti ozbiljnost i uticaj grešaka u podacima. Nisu sve greške jednake, i odgovarajući tok akcije može varirati ovisno o ozbiljnosti problema.

Manje nedosljednosti ili problemi s formatiranjem mogu biti pogodni za automatsku korekciju. Na primjer, samozacjeljujući radnik podataka zadužen za popravljanje neispravnog JSON-a može riješiti nedostajuće zareze ili neeskejpovane dvostruke navodnike bez značajnog mijenjanja značenja ili strukture podataka. Ove vrste grešaka se često mogu jednostavno ispraviti i imaju minimalan uticaj na ukupni integritet podataka.

Međutim, ozbiljnije greške koje fundamentalno mijenjaju značenje ili integritet podataka mogu zahtijevati drugačiji pristup. U takvim slučajevima, automatske korekcije možda neće biti dovoljne, i ljudska intervencija može biti neophodna kako bi se osigurala tačnost i validnost podataka.

Ovdje dolazi do izražaja koncept korištenja same vještačke inteligencije za određivanje ozbiljnosti grešaka. Koristeći mogućnosti AI modela, možemo dizajnirati samozacjeljujuće radnike za podatke koji ne samo da ispravljaju greške, već i procjenjuju ozbiljnost tih grešaka i donose informisane odluke o načinu njihovog rješavanja.

Na primjer, razmotrimo samozacjeljujućeg radnika za podatke zaduženog za ispravljanje nedosljednosti u podacima koji se unose u bazu podataka klijenata. Radnik može biti dizajniran da analizira podatke i identifikuje potencijalne greške, kao što su nedostajuće ili konfliktne informacije. Međutim, umjesto da automatski ispravlja sve greške, radnik može biti opremljen dodatnim pozivima alata koji mu omogućavaju da označi ozbiljne greške za pregled od strane ljudi.

Evo primjera kako se ovo može implementirati:

```ruby
class CustomerDataReviewer
  include Raix::ChatCompletion
  include Raix::FunctionDeclarations

  attr_accessor :customer

  function :flag_for_review, reason: { type: "string" } do |params|
    AdminNotifier.review_request(customer, params[:reason])
  end

  def initialize(customer)
    self.customer = customer
  end

  def call(customer_data)
    transcript << {
      system: "You are a customer data reviewer. Your task is to identify
        and correct inconsistencies in customer data.

        < additional instructions here... >

        If you encounter severe errors that require human review, use the
        `flag_for_review` tool to flag the data for manual intervention." }

    transcript << { user: customer.to_json }
    transcript << { assistant: "Reviewed/corrected data:\n```json\n" }

    self.stop = ["```"]

    chat_completion(json: true).then do |result|
      return if result.blank?

      customer.update(result)
    end
  end
end
```

U ovom primjeru, `CustomerDataHealer` radnik je dizajniran da identifikuje i ispravi nedosljednosti u korisničkim podacima. Još jednom, koristimo Ograđivanje odgovora i Ventriloquist za dobijanje strukturiranog izlaza. Važno je napomenuti da sistemska

direktiva radnika uključuje instrukcije za korištenje funkcije `flag_for_review` ako se naiđe na ozbiljne greške.

Kada radnik obrađuje korisničke podatke, analizira podatke i pokušava ispraviti sve nedosljednosti. Ako radnik utvrdi da su greške ozbiljne i zahtijevaju ljudsku intervenciju, može koristiti alat `flag_for_review` da označi podatke i navede razlog za označavanje.

Metoda `chat_completion` se poziva sa `json: true` da bi se ispravljeni korisnički podaci analizirali kao JSON. Ne postoji mogućnost ponavljanja nakon poziva funkcije, tako da će rezultat biti prazan ako je pozvana funkcija `flag_for_review`. U suprotnom, podaci o korisniku se ažuriraju pregledanim i potencijalno ispravljenim podacima.

Uključivanjem procjene ozbiljnosti grešaka i opcije označavanja podataka za ljudski pregled, radnik za samooporavljanje podataka postaje inteligentniji i prilagodljiviji. Može automatski rješavati manje greške dok ozbiljnije greške prosljeđuje ljudskim stručnjacima za ručnu intervenciju.

Specifični kriteriji za određivanje ozbiljnosti grešaka mogu se definisati u direktivi radnika na osnovu domenskog znanja i poslovnih zahtjeva. Faktori kao što su uticaj na integritet podataka, mogućnost gubitka ili oštećenja podataka, te posljedice netačnih podataka mogu se uzeti u obzir prilikom procjene ozbiljnosti.

Korištenjem vještačke inteligencije za procjenu ozbiljnosti grešaka i pružanjem opcija za ljudsku intervenciju, tehnike samooporavljanja podataka mogu postići ravnotežu između automatizacije i održavanja tačnosti podataka. Ovaj pristup osigurava da se manje greške efikasno ispravljaju dok ozbiljne greške dobijaju potrebnu pažnju i stručnost od ljudskih pregledača.

Kompleksnost domene

Kada razmatramo primjenu tehnika samooporavljanja podataka, važno je procijeniti kompleksnost domene podataka i pravila koja regulišu njenu strukturu i odnose. Kom-

pleksnost domene može značajno uticati na efektivnost i izvodljivost automatizovanih pristupa ispravljanju podataka.

Tehnike samooporavljanja podataka dobro funkcionišu kada podaci prate jasno definisane obrasce i ograničenja. U domenama gdje je struktura podataka relativno jednostavna i gdje su odnosi između elemenata podataka jednostavni, automatske korekcije se mogu primijeniti s visokim stepenom pouzdanosti. Na primjer, ispravljanje problema s formatiranjem ili primjena osnovnih ograničenja tipova podataka često se može efikasno riješiti pomoću radnika za samooporavljanje podataka.

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

Razmotrimo primjer složene domene: sistem za finansijsko trgovanje. U ovoj domeni, podaci uključuju različite finansijske instrumente, tržišne podatke, pravila trgovanja i regulatorne zahtjeve. Odnosi između različitih elemenata podataka mogu biti složeni, a pravila koja regulišu validnost i konzistentnost podataka mogu biti vrlo specifična za domenu.

U tako složenoj domeni, radnik za samooporavljanje 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 tržišni propisi, ograničenja trgovanja, izračuni rizika i procedure poravnanja. Automatske korekcije u ovom kontekstu možda neće uvijek obuhvatiti punu kompleksnost domene i mogu nenamjerno uvesti greške ili prekršiti pravila specifična za domenu.

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

1. **Obuka specifična za domenu**: AI modeli koji se koriste za samooporavljanje

podataka mogu biti usmjereni ili čak fino podešeni na skupovima podataka specifičnim za domenu koji obuhvataju složenosti i pravila određene domene. Izlažući modele reprezentativnim podacima i scenarijima, oni mogu naučiti obrasce, ograničenja i izuzetke specifične za domenu.

2. **Ograničenja zasnovana na pravilima**: Radnici za samooporavljanje podataka mogu se proširiti eksplicitnim ograničenjima zasnovanim na pravilima koja kodiraju znanje specifično za domenu. Ova pravila mogu definisati domenski stručnjaci i integrisati ih u proces ispravljanja podataka. AI modeli tada mogu koristiti ova pravila za usmjeravanje svojih odluka i osiguravanje usklađenosti sa zahtjevima specifičnim za domenu.
3. **Saradnja sa domenskim stručnjacima**: U složenim domenama, ključno je uključiti domenske stručnjake u dizajn i razvoj tehnika samooporavljanja podataka. Domenski stručnjaci mogu pružiti vrijedne uvide u složenosti podataka, poslovna pravila i potencijalne granične slučajeve. Njihovo znanje se može ugraditi u AI modele i radnike kako bi se poboljšala tačnost i pouzdanost automatskih ispravki podataka koristeći obrasce Čovjek u petlji.
4. **Inkrementalni i iterativni pristup**: Kada se radi sa složenim domenama, često je korisno usvojiti inkrementalni i iterativni pristup samooporavljanju podataka. Umjesto pokušaja automatizacije ispravki za cijelu domenu odjednom, fokusirajte se na specifične poddomene ili kategorije podataka gdje su pravila i ograničenja dobro shvaćena. Postepeno proširujte opseg tehnika samooporavljanja kako raste razumijevanje domene i tehnike se pokažu efikasnim.

Uzimajući u obzir složenost domene podataka i ugrađivanje domenskog znanja u tehnike samozacjeljujućih podataka, možete postići ravnotežu između automatizacije i tačnosti. Važno je prepoznati da samozacjeljujući podaci nisu univerzalno rješenje i da pristup treba prilagoditi specifičnim zahtjevima i izazovima svake domene.

U složenim domenama, hibridni pristup koji kombinuje tehnike samozacjeljujućih podataka sa ljudskom ekspertizom i nadzorom može biti najefikasniji. Automatske korekcije mogu se baviti rutinskim i dobro definisanim slučajevima, dok se složeni scenariji

ili izuzeci mogu označiti za ljudski pregled i intervenciju. Ovaj kolaborativni pristup osigurava da se ostvare prednosti automatizacije uz održavanje potrebne kontrole i tačnosti u složenim domenama podataka.

Objašnjivost i Transparentnost

Objašnjivost se odnosi na sposobnost razumijevanja i interpretacije rasuđivanja iza odluka koje donose AI modeli, dok transparentnost uključuje pružanje jasne vidljivosti u proces korekcije podataka.

U mnogim kontekstima, izmjene podataka moraju biti provjerljive i opravdane. Zainteresovane strane, 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 važno u domenama gdje tačnost i integritet podataka imaju značajne implikacije, kao što su finansije, zdravstvo i pravna pitanja.

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

1. **Lanac Razmišljanja**: Traženje od modela da objasni svoje razmišljanje "naglas" prije primjene promjena na podatke može omogućiti lakše razumijevanje procesa donošenja odluka i može generisati objašnjenja razumljiva ljudima za napravljene korekcije. Kompromis je malo više složenosti u odvajanju objašnjenja od strukturiranog izlaza podataka, što se može riješiti...
2. **Generisanje Objašnjenja**: Radnici za samozacjeljivanje podataka mogu biti opremljeni sposobnošću generisanja objašnjenja razumljivih ljudima za korekcije koje prave. Ovo se može postići traženjem od modela da prikaže svoj proces donošenja odluka kao lako razumljiva objašnjenja *integrisana u same podatke.* Na primjer, radnik za samozacjeljivanje podataka mogao bi generisati izvještaj koji ističe specifične nedosljednosti u podacima koje je identificirao, korekcije koje je primijenio i obrazloženje iza tih korekcija.

3. **Važnost Karakteristika**: AI modeli mogu biti upućeni s informacijama o važnosti različitih karakteristika ili atributa u procesu korekcije podataka kao dio njihovih direktiva. Te direktive, zauzvrat, mogu biti izložene ljudskim zainteresovanim stranama. Identifikovanjem ključnih faktora koji utiču na odluke modela, zainteresovane strane mogu steći uvid u rasuđivanje iza korekcija i procijeniti njihovu validnost.
4. **Bilježenje i Revizija**: Implementacija sveobuhvatnih mehanizama bilježenja i revizije je ključna za održavanje transparentnosti u procesu samozacjeljujućih podataka. Svaka korekcija podataka koju naprave AI modeli treba biti zabilježena, uključujući originalne podatke, korigovane podatke i specifične preduzete akcije. Ovaj revizijski trag omogućava retrospektivnu analizu i pruža jasan zapis o izmjenama napravljenim na podacima.
5. **Pristup sa Čovjekom u Petlji**: Uključivanje pristupa sa čovjekom u petlji može poboljšati objašnjivost i transparentnost tehnika samozacjeljujućih podataka. Uključivanjem ljudskih stručnjaka u pregled i validaciju AI-generisanih korekcija, organizacije mogu osigurati da su korekcije usklađene sa domenskim znanjem i poslovnim zahtjevima. Ljudski nadzor dodaje dodatni sloj odgovornosti i omogućava identifikaciju potencijalnih pristrasnosti ili grešaka u AI modelima.
6. **Kontinuirano Praćenje i Evaluacija**: Redovno praćenje i evaluacija performansi tehnika samozacjeljujućih podataka je esencijalno za održavanje transparentnosti i povjerenja. Procjenom tačnosti i efektivnosti AI modela tokom vremena, organizacije mogu identificirati bilo kakva odstupanja ili anomalije i preduzeti korektivne akcije. Kontinuirano praćenje pomaže osigurati da proces samozacjeljujućih podataka ostane pouzdan i usklađen sa željenim ishodima.

Objašnjivost i transparentnost su kritični aspekti pri implementaciji tehnika samozacjeljujućih podataka. Pružanjem jasnih objašnjenja za korekcije podataka, održavanjem sveobuhvatnih revizijskih tragova i uključivanjem ljudskog nadzora, organizacije mogu izgraditi povjerenje u proces samozacjeljujućih 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. Dok tehnike samozacjeljujućih podataka mogu značajno poboljšati kvalitet podataka i efikasnost, one ne bi trebale doći po cijenu gubitka vidljivosti i kontrole nad procesom korekcije podataka. Dizajniranjem radnika za samozacjeljivanje podataka s fokusom na objašnjivost i transparentnost, organizacije mogu iskoristiti snagu AI-ja uz održavanje potrebnog nivoa odgovornosti i povjerenja u podatke.

Nenamjerne Posljedice

Dok tehnike samozacjeljujućih podataka imaju za cilj poboljšanje kvaliteta i konzistentnosti podataka, ključno je biti svjestan potencijala za nenamjerne posljedice. Automatske korekcije, ako nisu pažljivo dizajnirane i praćene, mogu nenamjerno izmijeniti značenje ili kontekst podataka, što dovodi do naknadnih problema.

Jedan od primarnih rizika samozacjeljujućih podataka je uvođenje pristrasnosti ili grešaka u proces korekcije podataka. AI modeli, kao i bilo koji drugi softverski sistem, mogu biti podložni pristrasnostima prisutnim u podacima za treniranje ili uvedenim kroz dizajn algoritama. Ako se ove pristrasnosti ne identifikuju i ne ublaže, mogu se propagirati kroz proces samozacjeljujućih podataka i rezultirati iskrivljenim ili netačnim modifikacijama podataka.

Na primjer, razmotrimo samozacjeljujućeg radnika za podatke zaduženog za ispravljanje nedosljednosti u demografskim podacima kupaca. Ako je AI model naučio pristrasnosti iz historijskih podataka, kao što je povezivanje određenih zanimanja ili nivoa prihoda sa specifičnim spolovima ili etničkim pripadnostima, može donijeti pogrešne pretpostavke i modificirati podatke na način koji pojačava te pristrasnosti. Ovo može dovesti do netačnih profila kupaca, pogrešnih poslovnih odluka i potencijalno diskriminirajućih ishoda.

Druga potencijalna neželjena posljedica je gubitak vrijednih informacija ili konteksta tokom procesa ispravljanja podataka. Tehnike samozacjeljujućih podataka često se

fokusiraju na standardizaciju i normalizaciju podataka kako bi se osigurala konzistentnost. Međutim, u nekim slučajevima, originalni podaci mogu sadržavati nijanse, izuzetke ili kontekstualne informacije koje su važne za razumijevanje cjelokupne slike. Automatske korekcije koje slijepo provode standardizaciju mogu nenamjerno ukloniti ili zamaskirati ove vrijedne informacije.

Na primjer, zamislite samozacjeljujućeg radnika za podatke odgovornog za ispravljanje nedosljednosti u medicinskim kartonima. Ako radnik naiđe na medicinsku historiju pacijenta s rijetkim stanjem ili neuobičajenim planom liječenja, može pokušati normalizirati podatke kako bi odgovarali uobičajenijem obrascu. Međutim, čineći to, može izgubiti specifične detalje i kontekst koji su ključni za tačno predstavljanje jedinstvene situacije pacijenta. Ovaj gubitak informacija može imati ozbiljne implikacije za brigu o pacijentu i medicinsko odlučivanje.

Da bi se ublažili rizici od neželjenih posljedica, ključno je zauzeti proaktivan pristup pri dizajniranju i implementaciji tehnika samozacjeljujućih podataka:

1. **Temeljito testiranje i validacija**: Prije implementacije samozacjeljujućih radnika za podatke u produkciji, ključno je temeljito testirati i validirati njihovo ponašanje u različitim scenarijima. Ovo uključuje testiranje s reprezentativnim skupovima podataka koji pokrivaju različite granične slučajeve, izuzetke i potencijalne pristrasnosti. Rigorozno testiranje pomaže u identifikaciji i rješavanju neželjenih posljedica prije nego što utječu na stvarne podatke.
2. **Kontinuirano praćenje i evaluacija**: Implementacija mehanizama za kontinuirano praćenje i evaluaciju je ključna za otkrivanje i ublažavanje neželjenih posljedica tokom vremena. Redovno pregledanje ishoda procesa samozacjeljujućih podataka, analiziranje uticaja na povezane sisteme i odlučivanje, te prikupljanje povratnih informacija od zainteresovanih strana može pomoći u identifikaciji štetnih efekata i potaknuti pravovremene korektivne akcije. Ako vaša organizacija ima operativne kontrolne ploče, vjerovatno je dobra ideja dodati jasno vidljive metrike povezane s automatiziranim promjenama

podataka. Dodavanje alarma povezanih s velikim odstupanjima od normalne aktivnosti promjene podataka je vjerojatno još bolja ideja!

3. **Ljudski nadzor i intervencija**: Održavanje ljudskog nadzora i mogućnosti intervencije u procesu samozacjeljujućih podataka je ključno. Iako automatizacija može značajno poboljšati efikasnost, važno je da ljudski stručnjaci pregledaju i validiraju korekcije koje su napravili AI modeli, posebno u kritičnim ili osjetljivim domenama. Ljudska prosudba i stručnost u domeni mogu pomoći u identifikaciji i rješavanju neželjenih posljedica koje se mogu pojaviti.

4. **Objašnjivi AI (XAI) i transparentnost**: Kao što je diskutovano u prethodnom pododjeljku, uključivanje tehnika objašnjivog AI-ja i osiguravanje transparentnosti u procesu samozacjeljujućih podataka može pomoći u ublažavanju neželjenih posljedica. Pružanjem jasnih objašnjenja za korekcije podataka i održavanjem sveobuhvatnih revizijskih zapisa, organizacije mogu bolje razumjeti i pratiti obrazloženje iza modifikacija koje su napravili AI modeli.
5. **Inkrementalni i iterativni pristup**: Usvajanje inkrementalnog i iterativnog pristupa samozacjeljujućim podacima može pomoći u minimiziranju rizika od neželjenih posljedica. Umjesto primjene automatskih korekcija na cijeli skup podataka odjednom, počnite s podskupom podataka i postepeno proširujte opseg kako se tehnike pokažu efektivnim i pouzdanim. Ovo omogućava pažljivo praćenje i prilagođavanje tokom vremena, smanjujući uticaj neželjenih posljedica.
6. **Saradnja i povratne informacije**: Uključivanje zainteresovanih strana iz različitih domena i podsticanje saradnje i povratnih informacija tokom procesa samozacjeljujućih podataka može pomoći u identifikaciji i rješavanju neželjenih posljedica. Redovno traženje inputa od stručnjaka iz domene, korisnika podataka i krajnjih korisnika može pružiti vrijedne uvide u stvarni uticaj korekcija 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 samozacjeljujućih

podataka uz minimiziranje potencijalnih štetnih efekata. Važno je pristupiti samozacjeljujućim podacima kao iterativnom i kolaborativnom procesu, kontinuirano prateći, evaluirajući i usavršavajući tehnike kako bi se osiguralo da su usklađene sa željenim ishodima i održavaju integritet i pouzdanost podataka.

Pri razmatranju upotrebe obrazaca samozacjeljujućih podataka, ključno je pažljivo evaluirati ove faktore i odvagnuti prednosti naspram potencijalnih rizika i ograničenja. U nekim slučajevima, hibridni pristup koji kombinuje automatske korekcije s ljudskim nadzorom i intervencijom može biti najprikladnije rješenje.

Također je vrijedno napomenuti da tehnike samozacjeljujućih podataka ne bi trebale biti posmatrane kao zamjena za robusnu validaciju podataka, sanitaciju unosa i mehanizme za rukovanje greškama. Ove temeljne prakse ostaju kritične za osiguravanje integriteta i sigurnosti podataka. Samozacjeljujući podaci bi trebali biti posmatrani kao komplementarni pristup koji može proširiti i poboljšati ove postojeće mjere.

Na kraju, odluka o korištenju obrazaca samozacjeljujućih podataka zavisi od specifičnih zahtjeva, ograničenja i prioriteta vaše aplikacije. Pažljivim razmatranjem gore navedenih faktora i njihovim usklađivanjem s ciljevima i arhitekturom vaše aplikacije, možete donijeti informisane odluke o tome kada i kako efikasno koristiti tehnike samozacjeljujućih podataka.

Kontekstualno generisanje sadržaja

Obrasci kontekstualnog generisanja sadržaja koriste snagu velikih jezičkih modela (VJM) za generisanje dinamičkog i kontekstualno specifičnog sadržaja unutar aplikacija. Ova kategorija obrazaca prepoznaje važnost isporuke personaliziranog i relevantnog sadržaja korisnicima na osnovu njihovih specifičnih potreba, preferenci, 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, kao što su preporuke za primarni sadržaj.

Obrasci kontekstualnog generisanja sadržaja mogu igrati ključnu ulogu u poboljšanju nivoa angažmana korisnika, pružanju prilagođenih iskustava i automatizaciji zadataka kreiranja sadržaja kako za vas tako i za vaše korisnike. Koristeći obrasce koje opisujemo u ovom poglavlju, možete kreirati aplikacije koje dinamički generišu sadržaj, prilagođavajući se kontekstu i ulaznim podacima u realnom vremenu.

Obrasci funkcionišu integrisanjem VJM-ova u izlaze aplikacije, od korisničkog interfejsa (ponekad nazvanog "chrome"), do e-mailova i drugih oblika obavještenja, kao i svih protoka generisanja sadržaja.

Kada korisnik komunicira s aplikacijom ili pokrene specifični zahtjev za sadržajem, aplikacija bilježi relevantni kontekst, kao što su korisničke preference, prethodne interakcije ili specifični upiti. Ove kontekstualne informacije se zatim prosljeđuju u VJM, 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 generisan.

Sadržaj generisan putem VJM-a može poprimiti različite oblike, kao što su personalizirane preporuke, dinamički opisi proizvoda, prilagođeni e-mail odgovori, pa čak i cijeli članci ili blog objave. Jedna od najradikalnijih upotreba ovog sadržaja koju sam započeo prije više od godinu dana je dinamičko generisanje UI elemenata poput oznaka obrazaca, opisa alata i drugih vrsta objašnjavajućeg teksta.

Personalizacija

Jedna od ključnih prednosti obrazaca kontekstualnog generisanja sadržaja je mogućnost pružanja visoko personaliziranih iskustava korisnicima. Generisanjem sadržaja na osnovu konteksta specifičnog za korisnika, ovi obrasci omogućavaju aplikacijama da prilagode sadržaj individualnim interesima, preferencama i interakcijama korisnika.

Personalizacija nadilazi jednostavno umetanje korisničkog imena u generički sadržaj. Ona uključuje korištenje bogatog konteksta dostupnog o svakom korisniku za gener-

isanje sadržaja koji rezonira s njihovim specifičnim potrebama i željama. Ovaj kontekst može uključivati širok spektar faktora, kao što su:

1. **Informacije korisničkog profila**: Na najopćenitijem nivou primjene ove tehnike, demografski podaci, interesi, preference i drugi atributi profila mogu se koristiti za generisanje sadržaja koji se usklađuje s korisnikovom pozadinom i karakteristikama.
2. **Podaci o ponašanju**: Prethodne interakcije korisnika s aplikacijom, kao što su pregledane stranice, kliknuti linkovi ili kupljeni proizvodi, mogu pružiti vrijedne uvide u njihovo ponašanje i interese. Ovi podaci se mogu koristiti za generisanje prijedloga sadržaja koji odražava njihove obrasce angažmana i predviđa njihove buduće potrebe.
3. **Kontekstualni faktori**: Trenutni kontekst korisnika, kao što su njihova lokacija, uređaj, doba dana, pa čak i vremenske prilike, mogu utjecati na proces generisanja sadržaja. Na primjer, turistička aplikacija bi mogla imati AI radnika koji je u stanju generisati personalizirane preporuke na osnovu trenutne lokacije korisnika i prevladavajućih vremenskih uslova.

Korištenjem ovih kontekstualnih faktora, obrasci kontekstualnog generisanja sadržaja omogućavaju aplikacijama da isporuče sadržaj koji djeluje kao da je posebno napravljen za svakog pojedinačnog korisnika. Ovaj nivo personalizacije ima nekoliko značajnih prednosti:

1. **Povećani angažman**: Personalizirani sadržaj privlači pažnju korisnika i održava njihovu angažovanost s aplikacijom. Kada korisnici osjećaju da je sadržaj relevantan i direktno govori njihovim potrebama, vjerovatnije je da će provesti više vremena u interakciji s aplikacijom i istraživanju njenih funkcionalnosti.
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-commerce ili marketinških aplikacija, personalizirani sadržaj može značajno utjecati na stope konverzije. Predstavljanjem korisnicima proizvoda, ponuda ili preporuka koje su prilagođene njihovim preferencama i ponašanju, aplikacija može povećati vjerovatnoću da će korisnici poduzeti željene akcije, kao što su kupovina ili registracija za uslugu.

Produktivnost

Obrasci kontekstualnog generisanja sadržaja mogu značajno povećati određene vrste produktivnosti smanjujući potrebu za ručnim generisanjem sadržaja i uređivanjem u kreativnim procesima. Korištenjem snage VJM-ova, možete generisati visokokvalitetni sadržaj u velikim razmjerama, štedeći vrijeme i trud koji bi vaši kreatori sadržaja i programeri inače morali utrošiti na dosadan ručni rad.

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

Međutim, sa obrascima Kontekstualnog generisanja sadržaja, proces kreiranja sadržaja može biti uglavnom automatizovan. Veliki jezički modeli mogu generisati koherentan, gramatički ispravan i kontekstualno relevantan sadržaj na osnovu datih uputa i smjernica. Ova automatizacija nudi nekoliko prednosti za produktivnost:

1. **Smanjen ručni rad**: Delegiranjem zadataka generisanja sadržaja velikim jezičkim modelima, kreatori sadržaja se mogu fokusirati na zadatke višeg nivoa kao što su strategija sadržaja, ideacija i osiguranje kvaliteta. Oni mogu pružiti potreban kontekst, predloške i smjernice jezičkom modelu i pustiti ga da se bavi stvarnim generisanjem sadržaja. Ovo smanjuje ručni rad potreban za pisanje i uređivanje, omogućavajući kreatorima sadržaja da budu produktivniji i efikasniji.

2. **Brže kreiranje sadržaja**: Veliki jezički modeli mogu generisati sadržaj mnogo brže od ljudskih pisaca. Sa pravim uputama i smjernicama, jezički model može proizvesti više dijelova sadržaja u roku od nekoliko sekundi ili minuta. Ova brzina omogućava aplikacijama da generišu sadržaj mnogo brže, držeći korak sa zahtjevima korisnika i stalno promjenjivim digitalnim okruženjem.

> Da li brže kreiranje sadržaja vodi do situacije "tragedije zajedničkog dobra" gdje se internet guši u sadržaju koji niko ne čita. Nažalost, sumnjam da je odgovor da.

3. **Konzistentnost i kvalitet**: Veliki jezički modeli mogu trivijalno revidirati sadržaj tako da bude konzistentan u stilu, tonu i kvalitetu. Uz jasne smjernice i primjere, određene vrste aplikacija (tj. redakcije, PR, itd.) mogu osigurati da njihov sadržaj koji generišu ljudi bude usklađen s njihovim brendiranim glasom i zadovoljava željene standarde kvaliteta. Ova konzistentnost smanjuje potrebu za ekstenzivnim uređivanjem i revizijama, štedeći vrijeme i trud u procesu kreiranja sadržaja.
4. **Iteracija i optimizacija**: Obrasci Kontekstualnog generisanja sadržaja omogućavaju brzu iteraciju i optimizaciju sadržaja. Podešavanjem uputa, predložaka ili smjernica datih jezičkom modelu, vaše aplikacije mogu brzo generisati varijacije sadržaja i testirati različite pristupe na automatizovan način koji nikada ranije nije bio moguć. Ovaj iterativni proces omogućava brže eksperimentisanje i usavršavanje strategija sadržaja, što vremenom dovodi do efektivnijeg i angažovanijeg sadržaja. Ova konkretna tehnika može biti potpuna prekretnica za aplikacije kao što je e-trgovina koje žive i umiru na osnovu stopa napuštanja i angažmana

Važno je napomenuti da iako obrasci Kontekstualnog generisanja sadržaja mogu značajno povećati produktivnost, oni ne eliminišu potpuno potrebu za ljudskim učešćem. Kreatori sadržaja i urednici i dalje igraju ključnu ulogu u definisanju sveukupne strategije sadržaja, pružanju smjernica jezičkom modelu i osiguravanju kvaliteta i prikladnosti generisanog sadržaja.

Automatizacijom više repetitivnih i vremenski zahtjevnih aspekata kreiranja sadržaja, obrasci Kontekstualnog generisanja sadržaja oslobađaju dragocjeno ljudsko vrijeme i resurse koji se mogu preusmjeriti na zadatke više vrijednosti. Ova povećana produktivnost vam omogućava da isporučite više personalizovanog i angažujućeg sadržaja korisnicima dok optimizujete radne tokove kreiranja sadržaja.

Brza iteracija i eksperimentisanje

Obrasci Kontekstualnog generisanja sadržaja vam omogućavaju da brzo iterirate i eksperimentišete sa različitim varijacijama sadržaja, omogućavajući bržu optimizaciju i usavršavanje vaše strategije sadržaja. Možete generisati više verzija sadržaja u roku od nekoliko sekundi, jednostavno podešavanjem konteksta, predložaka ili smjernica datih modelu.

Ova mogućnost brze iteracije nudi nekoliko ključnih prednosti:

1. **Testiranje i optimizacija**: Sa mogućnošću brzog generisanja varijacija sadržaja, možete lako testirati različite pristupe i mjeriti njihovu efektivnost. Na primjer, možete generisati više verzija opisa proizvoda ili marketinške poruke, svaku prilagođenu specifičnom segmentu korisnika ili kontekstu. Analiziranjem metrika angažmana korisnika, kao što su stope klikova ili stope konverzije, možete identifikovati najefikasnije varijacije sadržaja i optimizovati vašu strategiju sadržaja u skladu s tim.

2. **A/B testiranje**: Obrasci Kontekstualnog generisanja sadržaja omogućavaju besprijekorno A/B testiranje sadržaja. Možete generisati dvije ili više varijacija sadržaja i nasumično ih servirati različitim grupama korisnika. Upoređivanjem performansi svake varijacije, možete odrediti koji sadržaj najbolje rezonuje sa vašom ciljnom publikom. Ovaj pristup vođen podacima vam omogućava da donosite informisane odluke i kontinuirano usavršavate svoj sadržaj kako biste maksimizirali angažman korisnika i postigli željene rezultate.
3. **Eksperimenti personalizacije**: Brza iteracija i eksperimentisanje su posebno vrijedni kada je u pitanju personalizacija. Sa obrascima Kontekstualnog generisanja sadržaja, možete brzo generisati personalizovane varijacije sadržaja zasnovane na različitim segmentima korisnika, preferencijama ili ponašanjima. Eksperimentisanjem sa različitim strategijama personalizacije, možete identifikovati najefikasnije pristupe za angažovanje pojedinačnih korisnika i isporuku prilagođenih iskustava.
4. **Prilagođavanje promjenljivim trendovima**: Mogućnost brze iteracije i eksperimentisanja omogućava vam da ostanete agilni i prilagodite se promjenljivim trendovima i preferencijama korisnika. Kako se pojavljuju nove teme, ključne riječi ili ponašanja korisnika, možete brzo generisati sadržaj koji je usklađen s tim trendovima. Kontinuiranim eksperimentisanjem i usavršavanjem vašeg sadržaja, možete ostati relevantni i održati konkurentsku prednost u digitalnom okruženju koje se neprestano razvija.
5. **Ekonomično eksperimentisanje**: Tradicionalno eksperimentisanje sa sadržajem često zahtijeva značajno vrijeme i resurse, jer kreatori sadržaja moraju ručno razvijati i testirati različite varijacije. Međutim, sa obrascima kontekstualnog generisanja sadržaja, trošak eksperimentisanja je značajno smanjen. Veliki jezički modeli mogu brzo generisati varijacije sadržaja u većem obimu, omogućavajući vam da istražite širok spektar ideja i pristupa bez značajnih troškova.

Za maksimalno iskorištavanje brze iteracije i eksperimentisanja, važno je imati dobro definisan okvir za eksperimentisanje. Ovaj okvir bi trebao uključivati:

- Jasne ciljeve i hipoteze za svaki eksperiment
- Odgovarajuće metrike i mehanizme praćenja za mjerenje performansi sadržaja
- Strategije segmentacije i ciljanja kako bi se osiguralo da odgovarajuće varijacije sadržaja budu dostavljene pravim korisnicima
- Alate za analizu i izvještavanje za izvlačenje uvida iz eksperimentalnih podataka
- Proces za ugrađivanje naučenog i optimizacija u vašu strategiju sadržaja

Prihvatanjem brze iteracije i eksperimentisanja, možete kontinuirano usavršavati i optimizirati vaš sadržaj, osiguravajući da ostane angažovan, relevantan i efikasan u postizanju ciljeva vaše aplikacije. Ovaj agilni pristup kreiranju sadržaja omogućava vam da ostanete korak ispred i pružite izuzetno korisničko iskustvo.

Skalabilnost i efikasnost

Kako aplikacije rastu i potražnja za personaliziranim sadržajem se povećava, obrasci kontekstualnog generisanja sadržaja omogućavaju efikasno skaliranje kreiranja sadržaja. Veliki jezički modeli mogu istovremeno generisati sadržaj za veliki broj korisnika i konteksta, bez potrebe za proporcionalnim povećanjem ljudskih resursa. Ova skalabilnost omogućava aplikacijama da pruže personalizirana iskustva rastućoj bazi korisnika bez opterećenja njihovih mogućnosti kreiranja sadržaja.

Imajte na umu da se kontekstualno generisanje sadržaja može efikasno koristiti za internacionalizaciju vaše aplikacije "u hodu". Zapravo, to je tačno ono što sam uradio koristeći svoj Instant18n Gem za isporuku Olympije na više od pola desetine jezika, iako nismo ni godinu dana stari.

AI pogonjena lokalizacija

Ako mi dozvolite da se pohvalim na trenutak, mislim da je moja Instant18n biblioteka za Rails aplikacije revolucionarni primjer obrasca "Kontekstualnog generisanja sadržaja" u

akciji, pokazujući transformativni potencijal AI-ja u razvoju aplikacija. Ovaj gem koristi snagu OpenAI-jevog GPT velikog jezičkog modela da revolucionizira način na koji se internacionalizacija i lokalizacija tretiraju u Rails aplikacijama.

Tradicionalno, internacionalizacija Rails aplikacije uključuje ručno definisanje ključeva za prevođenje i pružanje odgovarajućih prevoda za svaki podržani jezik. Ovaj proces može biti dugotrajan, resursno intenzivan i sklon nedosljednostima. Međutim, sa Instant18n gemom, paradigma lokalizacije je potpuno redefinisana.

Integracijom velikog jezičkog modela, Instant18n gem vam omogućava da generirate prevode u hodu, na osnovu konteksta i značenja teksta. Umjesto oslanjanja na unaprijed definisane ključeve za prevođenje i statičke prevode, gem dinamički prevodi tekst koristeći snagu AI-ja. Ovaj pristup nudi nekoliko ključnih prednosti:

1. **Besprijekorna lokalizacija**: Sa Instant18n gemom, programeri više ne moraju ručno definisati i održavati datoteke za prevođenje za svaki podržani jezik. Gem automatski generira prevode na osnovu dostavljenog teksta i željenog ciljnog jezika, čineći proces lokalizacije jednostavnim i besprijekornim.
2. **Kontekstualna tačnost**: AI-ju se može dati dovoljno konteksta da shvati nijanse teksta koji se prevodi. Može uzeti u obzir okolni kontekst, idiome i kulturološke reference kako bi generisao prevode koji su tačni, prirodni i kontekstualno prikladni.
3. **Opsežna jezička podrška**: Instant18n gem koristi ogromno znanje i lingvističke sposobnosti GPT-a, omogućavajući prevode na širok spektar jezika. Od uobičajenih jezika poput španskog i francuskog do opskurnijih ili izmišljenih jezika poput klingonskog i vilinskog, gem može zadovoljiti širok spektar prevodilačkih zahtjeva.
4. **Fleksibilnost i kreativnost**: Gem prevazilazi tradicionalne jezičke prevode i omogućava kreativne i nekonvencionalne opcije lokalizacije. Programeri mogu prevoditi tekst u različite stilove, dijalekte, pa čak i izmišljene jezike, otvarajući nove mogućnosti za jedinstvena korisnička iskustva i angažujući sadržaj.

5. **Optimizacija performansi**: Instant18n gem uključuje mehanizme keširanja za poboljšanje performansi i smanjenje opterećenja kod ponovljenih prevoda. Prevedeni tekst se kešira, omogućavajući da se naknadni zahtjevi za istim prevodom brzo serviraju bez potrebe za redundantnim API pozivima.

Instant18n gem predstavlja primjer snage obrasca "Kontekstualnog generisanja sadržaja" korištenjem AI-ja za dinamičko generisanje lokaliziranog sadržaja. On pokazuje kako se AI može integrisati u osnovnu funkcionalnost Rails aplikacije, transformišući način na koji programeri pristupaju internacionalizaciji i lokalizaciji.

Eliminiranjem potrebe za ručnim upravljanjem prijevodima i omogućavanjem prijevoda u realnom vremenu na osnovu konteksta, Instant18n gem štedi programerima značajno vrijeme i trud. Omogućava im da se fokusiraju na izgradnju osnovnih funkcionalnosti svoje aplikacije, istovremeno osiguravajući da se aspekt lokalizacije odvija besprijekorno i precizno.

Važnost korisničkog testiranja i povratnih informacija

Konačno, uvijek imajte na umu važnost korisničkog testiranja i povratnih informacija. Ključno je potvrditi da kontekstualno generiranje sadržaja ispunjava očekivanja korisnika i usklađeno je s ciljevima aplikacije. Kontinuirano unapređujte i usavršavajte generirani sadržaj na osnovu korisničkih uvida i analitike. Ako generirate dinamički sadržaj u velikom obimu koji bi bilo nemoguće ručno validirati od strane vas i vašeg tima, razmotrite dodavanje mehanizama za povratne informacije koji omogućavaju korisnicima da prijave sadržaj koji je čudan ili pogrešan, zajedno s objašnjenjem zašto. Te dragocjene povratne informacije mogu čak biti proslijeđene AI radniku zaduženom za prilagođavanje komponente koja je generirala sadržaj!

Generative UI

Pažnja je danas toliko dragocjena da efektivno korisničko angažovanje sada zahtijeva softverska iskustva koja nisu samo besprijekorna i intuitivna, već i izrazito personalizovana prema individualnim potrebama, preferencijama i kontekstima. Kao rezultat toga, dizajneri i programeri sve češće se suočavaju s izazovom kreiranja korisničkih sučelja koja se mogu prilagoditi i odgovoriti jedinstvenim zahtjevima svakog korisnika *u većem obimu*.

Generative UI (GenUI) je zaista revolucionarni pristup dizajnu korisničkog sučelja koji koristi snagu velikih jezičkih modela (LLM) za stvaranje visoko personalizovanih i dinamičnih korisničkih iskustava u realnom vremenu. Želio sam vam barem dati uvod u GenUI u ovoj knjizi, jer vjerujem da je to jedna od najotvorenijih prilika koje trenutno postoje u području dizajna aplikacija i frameworka. Uvjeren sam da će se u ovoj posebnoj niši pojaviti desetine ili više novih uspješnih komercijalnih i projekata otvorenog koda.

U svojoj srži, GenUI kombinuje principe Kontekstualnog generiranja sadržaja s naprednim AI tehnikama za dinamičko generisanje elemenata korisničkog sučelja, kao što su tekst, slike i rasporedi, na osnovu dubokog razumijevanja korisničkog konteksta, preferencija i ciljeva. GenUI omogućava dizajnerima i programerima da kreiraju sučelja koja se prilagođavaju i razvijaju kao odgovor na korisničke interakcije, pružajući nivo personalizacije koji ranije nije bio dostižan.

GenUI predstavlja fundamentalnu promjenu u načinu na koji pristupamo dizajnu korisničkog sučelja. Umjesto dizajniranja za mase, GenUI nam omogućava da dizajniramo za pojedinca. Personalizirani sadržaj i sučelja imaju potencijal stvaranja korisničkih iskustava koja rezoniraju sa svakim korisnikom na dubljem nivou, povećavajući angažman, zadovoljstvo i lojalnost.

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

Uprkos izazovima, personalizovana iskustva u većem obimu imaju moć da potpuno transformišu način na koji komuniciramo s digitalnim proizvodima i uslugama. To otvara mogućnosti za stvaranje inkluzivnih i pristupačnih sučelja koja odgovaraju različitim potrebama korisnika, bez obzira na njihove sposobnosti, porijeklo ili preferencije.

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

Generisanje tekstualnog sadržaja za korisnička sučelja

Tekstualni elementi koji postoje u elementima korisničkog sučelja vaše aplikacije, kao što su oznake obrazaca, oblačići s objašnjenjima i opisni tekst, obično su hardkodirani u predloške ili UI komponente, pružajući konzistentno ali generičko iskustvo za sve korisnike. Koristeći obrasce kontekstualnog generiranja sadržaja, možete transformisati ove statične elemente u dinamične, kontekstualno svjesne i personalizovane komponente.

Personalizovani obrasci

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

Međutim, važno je postići ravnotežu između personalizacije i konzistentnosti. Dok prilagođavanje obrazaca individualnim korisnicima može biti korisno, ključno je održati određeni nivo poznatosti i predvidljivosti. Korisnici bi i dalje trebali moći lako prepoznati i navigirati kroz obrasce, čak i s personalizovanim elementima.

Evo nekoliko ideja za personalizovane obrasce za 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 sačuvanim informacijama. Ovo ne samo da štedi vrijeme, već i pokazuje da aplikacija razumije i pamti korisničke preferencije.

Čekajte malo, zar nije ova tehnika nešto što bi se moglo uraditi bez upotrebe AI-ja? Naravno, ali ljepota pokretanja ovakve funkcionalnosti pomoću AI-ja je dvostruka: 1) koliko jednostavna može biti implementacija i 2) koliko otporna može biti dok se vaš UI mijenja i razvija tokom vremena.

Hajde da napravimo servis za naš teoretski sistem za obradu narudžbi, koji će pokušati proaktivno popuniti odgovarajuću adresu za dostavu za korisnika.

```
class OrderShippingAddressSubscriber
  include Raix::ChatCompletion

  attr_accessor :order

  delegate :customer, to: :order

  DIRECTIVE = "You are a smart order processing assistant. Given the
  customer's order history, guess the most likely shipping address
  for the current order."

  def order_created(order)
    return unless order.pending? && order.shipping_address.blank?

    self.order = order

    transcript.clear
    transcript << { system: DIRECTIVE }
    transcript << { user: "Order History: #{order_history.to_json}" }
    transcript << { user: "Current Order: #{order.to_json}" }

    response = chat_completion
    apply_predicted_shipping_address(order, response)
  end

  private

  def apply_predicted_shipping_address(order, response)
    # extract the shipping address from the response...
```

```
    # ...and assume there's some sort of live update of the address fields
    order.update(shipping_address:)
  end

  def order_history
    customer.orders.successful.limit(100).map do |order|
      {
        date: order.date,
        description: order.description,
        shipping_address: order.shipping_address
      }
    end
  end
end
```

Ovaj primjer je vrlo pojednostavljen, ali bi trebao raditi u većini slučajeva. Ideja je pustiti vještačku inteligenciju da nagađa na isti način kao što bi to učinio čovjek. Da bismo pojasnili o čemu govorim, razmotrimo neke probne podatke:

```
Order History:
[
  {"date": "2024-01-03", "description": "garden soil mix",
   "shipping_address": "123 Country Lane, Rural Town"},
  {"date": "2024-01-15", "description": "hardcover fiction novels",
   "shipping_address": "456 City Apt, Metroville"},
  {"date": "2024-01-22", "description": "baby diapers", "shipping_address":
   "789 Suburb St, Quietville"},
  {"date": "2024-02-01", "description": "organic vegetables",
   "shipping_address": "123 Country Lane, Rural Town"},
  {"date": "2024-02-17", "description": "mystery thriller book set",
   "shipping_address": "456 City Apt, Metroville"},
  {"date": "2024-02-25", "description": "baby wipes",
   "shipping_address": "789 Suburb St, Quietville"},
  {"date": "2024-03-05", "description": "flower seeds",
   "shipping_address": "123 Country Lane, Rural Town"},
  {"date": "2024-03-20", "description": "biographies",
   "shipping_address": "456 City Apt, Metroville"},
  {"date": "2024-03-30", "description": "baby formula",
   "shipping_address": "789 Suburb St, Quietville"},
  {"date": "2024-04-12", "description": "lawn fertilizer",
```

```
    "shipping_address": "123 Country Lane, Rural Town"},
  {"date": "2024-04-22", "description": "science fiction novels",
    "shipping_address": "456 City Apt, Metroville"},
  {"date": "2024-05-02", "description": "infant toys",
    "shipping_address": "789 Suburb St, Quietville"},
  {"date": "2024-05-14", "description": "outdoor grill",
    "shipping_address": "123 Country Lane, Rural Town"},
  {"date": "2024-05-29", "description": "literary classics",
  "shipping_address": "456 City Apt, Metroville"},
  {"date": "2024-06-11", "description": "baby clothes",
    "shipping_address": "789 Suburb St, Quietville"},
  {"date": "2024-07-01", "description": "watering can",
    "shipping_address": "123 Country Lane, Rural Town"},
  {"date": "2024-07-18", "description": "non-fiction essays",
  "shipping_address": "456 City Apt, Metroville"},
  {"date": "2024-07-28", "description": "baby bath items",
  "shipping_address": "789 Suburb St, Quietville"},
  {"date": "2024-08-09", "description": "herb garden kit",
  "shipping_address": "123 Country Lane, Rural Town"},
  {"date": "2024-08-24", "description": "children's books",
  "shipping_address": "456 City Apt, Metroville"}
]
```

Jeste li primjetili obrazac u podacima? Garantujem vam da je ovo jednostavna stvar za LLM. Da demonstriramo, pitajmo GPT-4 koja je najvjerovatnija 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 kao što je GPT-4 za ovaj zadatak, u pravu ste! Isprobao sam isti prompt na Mistral 7B Instruct i proizveo je sljedeći odgovor brzinom od 75 tokena po 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.
```

Da li su overhead i troškovi ove tehnike vrijedni toga da iskustvo kupovine učine magičnijim? Za mnoge online prodavače, apsolutno. I kako stvari stoje, troškovi AI računanja će samo padati, posebno za pružaoce usluga hostinga open source modela u trci prema dnu.

Koristite Predložak Promptova i StructuredIO zajedno sa Ograđivanjem Odgovora da optimizirate ovakvu vrstu chat završetka.

Adaptivno Redoslijed Polja

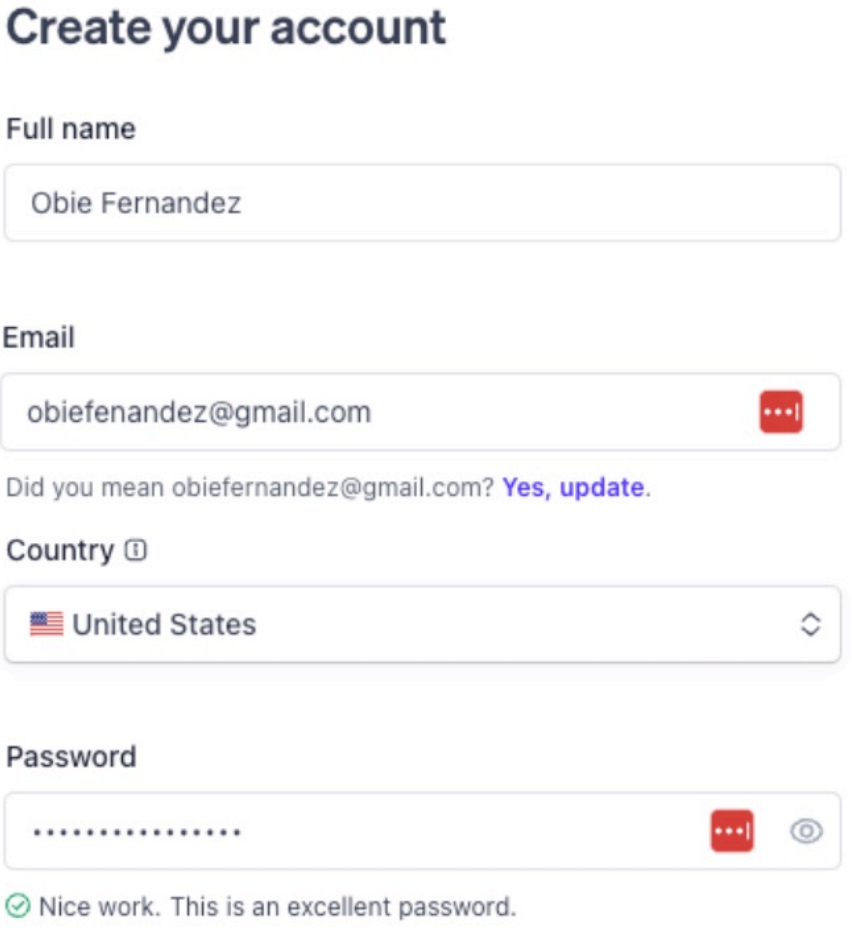
Redoslijed kojim se prezentiraju polja formulara može značajno uticati na korisničko iskustvo i stope završetka. Sa GenUI, možete dinamički prilagoditi redoslijed polja na osnovu korisničkog konteksta i važnosti svakog polja. Na primjer, ako korisnik popunjava registracijski formular za fitness aplikaciju, formular može dati prioritet poljima vezanim za njihove fitness ciljeve i preferencije, čineći proces relevantnijim i angažovanijim.

Personalizirani Mikrotekst

Instrukcijski tekst, poruke o greškama i drugi mikrotekst povezan s formularima također se može personalizirati koristeći GenUI. Umjesto prikazivanja generičkih poruka o greškama poput "Nevažeća email adresa," možete generirati korisnije i kontekstualne poruke kao što je "Molimo unesite važeću email adresu kako biste primili potvrdu vaše narudžbe." Ovi personalizirani detalji mogu učiniti iskustvo s formularom pristupačnijim i manje frustrirajućim.

Personalizirana Validacija

U skladu s Personaliziranim Mikrotekstom, mogli biste koristiti AI za validaciju formulara na načine koji djeluju magično. Zamislite da pustite AI da validira korisnički profilni formular, tražeći potencijalne greške na *semantičkom* nivou.



Slika 9. Možete li uočiti semantičku validaciju koja se događa?

Progresivno Otkrivanje

GenUI može inteligentno odrediti koja su polja formulara esencijalna na osnovu korisničkog konteksta i postepeno otkrivati dodatna polja prema potrebi. Ova tehnika progresivnog otkrivanja pomaže smanjiti kognitivno opterećenje i čini proces popunjavanja formulara upravljivijim. Na primjer, ako se korisnik prijavljuje za osnovnu

pretplatu, formular može inicijalno prikazati samo esencijalna polja, a kako korisnik napreduje ili odabire određene opcije, dodatna relevantna polja mogu se dinamički uvoditi.

Kontekstualno Svjestan Tekst za Objašnjenja

Tooltips se često koriste za pružanje dodatnih informacija ili smjernica korisnicima kada prelaze mišem preko ili komuniciraju sa specifičnim elementima. Sa pristupom "Kontekstualne Generacije Sadržaja", možete generirati tooltips koji se prilagođavaju korisničkom kontekstu i pružaju relevantne informacije. Na primjer, ako korisnik istražuje kompleksnu funkciju, tooltip može ponuditi personalizirane savjete ili primjere bazirane na njihovim prethodnim interakcijama ili nivou vještina.

Tekst za objašnjenja, kao što su instrukcije, opisi ili poruke za pomoć, može se dinamički generirati na osnovu korisničkog konteksta. Umjesto prezentiranja 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 za objašnjenja može pružiti personalizirane smjernice ili savjete za rješavanje problema.

Mikrotekst se odnosi na male dijelove teksta koji vode korisnike kroz vašu aplikaciju, kao što su oznake dugmadi, poruke o greškama ili upiti za potvrdu. Primjenom pristupa Kontekstualne Generacije Sadržaja na mikrotekst, možete kreirati adaptivni UI koji reaguje na korisničke akcije i pruža relevantan i koristan tekst. Na primjer, ako korisnik treba izvršiti kritičnu akciju, upit za potvrdu može se dinamički generirati kako bi pružio jasnu i personaliziranu poruku.

Personalizirani tekst za objašnjenja i tooltips mogu značajno poboljšati proces uvođenja novih korisnika. Pružanjem kontekstualno specifičnih smjernica i primjera, možete pomoći korisnicima da brzo razumiju i navigiraju aplikaciju, smanjujući krivulju učenja i povećavajući usvajanje.

Dinamički i kontekstualno svjesni chrome elementi također mogu učiniti aplikaciju intuitivnijom i angažovanijom. Korisnici će vjerojatnije komunicirati s funkcijama i istraživati ih kada je prateći tekst prilagođen njihovim specifičnim potrebama i interesima.

Do sada smo obradili ideje za unapređenje postojećih UI paradigmi pomoću vještačke inteligencije, ali šta je sa preispitivanjem načina na koji se korisnički interfejsi dizajniraju i implementiraju na radikalniji način?

Definisanje generativnog UI-ja

Za razliku od tradicionalnog UI dizajna, gdje dizajneri kreiraju fiksne, statične interfejse, GenUI nagovjještava budućnost u kojoj naš softver posjeduje fleksibilna, personalizovana iskustva koja se mogu razvijati i prilagođavati u realnom vremenu. Svaki put kada koristimo interfejs za razgovor pogonjen vještačkom inteligencijom, dozvoljavamo AI-ju da se prilagodi specifičnim potrebama korisnika. GenUI ide korak dalje primjenjujući taj nivo prilagodljivosti na *vizuelni* interfejs softvera.

Razlog zbog kojeg je danas moguće eksperimentisati sa GenUI idejama je taj što veliki jezički modeli već razumiju programiranje i njihovo osnovno znanje uključuje UI tehnologije i okvire. Pitanje je, dakle, mogu li se veliki jezički modeli koristiti za generisanje UI elemenata, kao što su tekst, slike, rasporedi, pa čak i cijeli interfejsi, koji su prilagođeni svakom pojedinačnom korisniku. Model bi mogao biti upućen da uzme u obzir različite faktore, kao što su prethodne interakcije korisnika, izražene preference, demografske informacije i trenutni kontekst upotrebe, kako bi kreirao visoko personalizovane i relevantne interfejse.

GenUI se razlikuje od tradicionalnog dizajna korisničkog interfejsa u nekoliko ključnih aspekata:

1. **Dinamičan i adaptivan**: Tradicionalni UI dizajn podrazumijeva kreiranje fiksnih, statičnih interfejsa koji ostaju isti za sve korisnike. Nasuprot tome, GenUI omogućava interfejse koji se mogu dinamički prilagođavati i mijenjati na osnovu potreba korisnika i konteksta. To znači da ista aplikacija može predstaviti različite interfejse različitim korisnicima, pa čak i istom korisniku u različitim situacijama.
2. **Personalizacija na velikoj skali**: Kod tradicionalnog dizajna, kreiranje personalizovanih iskustava za svakog korisnika često je nepraktično zbog potrebnog vremena i resursa. S druge strane, GenUI omogućava personalizaciju na velikoj skali. Koristeći AI, dizajneri mogu kreirati interfejse koji se automatski prilagođavaju jedinstvenim potrebama i preferencama svakog korisnika, bez potrebe za ručnim dizajniranjem i razvijanjem posebnih interfejsa za svaki segment korisnika.
3. **Fokus na rezultate**: Tradicionalni UI dizajn često se fokusira na kreiranje vizuelno privlačnih i funkcionalnih interfejsa. Iako su ovi aspekti i dalje važni u GenUI-ju, primarni fokus se premješta ka postizanju željenih korisničkih rezultata. GenUI teži stvaranju interfejsa koji su optimizovani za specifične ciljeve i zadatke svakog korisnika, dajući prioritet upotrebljivosti i efikasnosti nad čisto estetskim razmatranjima.
4. **Kontinuirano učenje i poboljšanje**: GenUI sistemi mogu kontinuirano učiti i poboljšavati se tokom vremena na osnovu korisničkih interakcija i povratnih informacija. Kako korisnici koriste generisane interfejse, AI modeli mogu prikupljati podatke o ponašanju korisnika, preferencama i rezultatima, koristeći ove informacije za usavršavanje i optimizaciju budućih generacija interfejsa. Ovaj iterativni proces učenja omogućava GenUI sistemima da vremenom postaju sve efikasniji u zadovoljavanju potreba korisnika.

Važno je napomenuti da GenUI nije isto što i AI-potpomognuti alati za dizajn, poput onih koji pružaju sugestije ili automatizuju određene zadatke dizajna. Iako ovi alati mogu biti korisni u pojednostavljivanju procesa dizajna, oni se i dalje oslanjaju na dizajnere da donose konačne odluke i kreiraju statične interfejse. GenUI, s druge

strane, podrazumijeva da AI sistem preuzima aktivniju ulogu u stvarnom generisanju i prilagođavanju interfejsa na osnovu korisničkih podataka i konteksta.

GenUI predstavlja značajan pomak u načinu na koji pristupamo dizajnu korisničkog interfejsa, udaljavanje od rješenja koja odgovaraju svima i približavanje visoko personalizovanim, adaptivnim iskustvima. Koristeći moć AI-ja, GenUI ima potencijal da revolucionizuje način na koji komuniciramo sa digitalnim proizvodima i uslugama, stvarajući interfejse koji su intuitivniji, angažovaniji i efikasniji za svakog pojedinačnog korisnika.

Primjer

Da bismo ilustrovali koncept GenUI-ja, razmotrimo hipotetičku fitness aplikaciju pod nazivom "FitAI". Ova aplikacija ima za cilj da pruži personalizovane planove vježbanja i savjete o ishrani korisnicima na osnovu njihovih individualnih ciljeva, nivoa kondicije i preferencija.

U tradicionalnom pristupu UI dizajnu, FitAI bi mogao imati fiksan set ekrana i elemenata koji su isti za sve korisnike. Međutim, sa GenUI-jem, interfejs aplikacije bi se mogao dinamički prilagođavati jedinstvenim potrebama i kontekstu svakog korisnika.

Ovaj pristup je pomalo teško zamisliti za implementaciju u 2024. godini i možda čak nema adekvatan povrat investicije, ali je moguć.

Evo kako bi to moglo funkcionisati:

1. **Uvođenje**:
    - Umjesto standardnog upitnika, FitAI koristi konverzacijsku vještačku inteligenciju za prikupljanje informacija o ciljevima korisnika, trenutnom nivou kondicije i preferencama.

- Na osnovu ove početne interakcije, AI generiše personalizovani raspored kontrolne table, naglašavajući funkcije i informacije koje su najrelevantnije za ciljeve korisnika.
- Trenutna AI tehnologija bi mogla imati na raspolaganju izbor komponenti ekrana za sastavljanje personalizovane kontrolne table.
- Buduća AI tehnologija bi mogla preuzeti ulogu iskusnog UI dizajnera i zapravo kreirati kontrolnu tablu *od nule*.

2. **Planer treninga**:

- AI prilagođava interfejs planera treninga specifično prema nivou iskustva korisnika i dostupnoj opremi.
- Za početnika bez opreme, mogao bi prikazivati jednostavne vježbe sa vlastitom težinom tijela uz detaljne upute i video zapise.
- Za naprednog korisnika s pristupom teretani, mogao bi prikazivati složenije rutine s manje objašnjenja.
- Sadržaj planera treninga se ne filtrira jednostavno iz velikog skupa podataka. Može se generisati *u trenutku* na osnovu baze znanja koja se pretražuje s kontekstom koji uključuje sve što se zna o korisniku.

3. **Praćenje napretka**:

- Interfejs za praćenje napretka razvija se na osnovu korisnikovih ciljeva i obrazaca angažmana.
- Ako je korisnik prvenstveno fokusiran na gubitak težine, interfejs bi mogao istaknuto prikazivati graf trenda težine i statistiku potrošnje kalorija.
- Za korisnika koji gradi mišićnu masu, mogao bi isticati 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 preći u režim u kojem 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 pokazivati prijedloge biljnih obroka i izvore proteina.
    - Za korisnika s netolerancijom na gluten, automatski bi filtrirala namirnice koje sadrže gluten iz preporuka.
    - Ponovno, sadržaj se ne izvlači iz masivnog nadskupa podataka o obrocima koji se odnosi na sve korisnike, već se sintetizira iz baze znanja koja sadrži informacije prilagodljive specifičnoj situaciji i ograničenjima korisnika.
    - Na primjer, recepti se generišu sa specifikacijama sastojaka koje odgovaraju konstantno promjenjivim kalorijskim potrebama korisnika kako se njihov nivo kondicije i tjelesne statistike razvijaju.
5. **Motivacijski elementi**:
    - Motivacijski sadržaj aplikacije i obavještenja su personalizovani na osnovu tipa ličnosti korisnika i reakcije na različite motivacijske strategije.
    - Neki korisnici mogu primati ohrabrujuće poruke, dok drugi dobijaju povratne informacije više zasnovane na podacima.

U ovom primjeru, GenUI omogućava FitAI-u da stvori visoko prilagođeno iskustvo za svakog korisnika, potencijalno povećavajući angažman, zadovoljstvo i vjerovatnoću postizanja fitness ciljeva. Elementi interfejsa, sadržaj, pa čak i "ličnost" aplikacije prilagođavaju se kako bi najbolje služili potrebama i preferencijama svakog pojedinačnog korisnika.

Prelazak na dizajn orijentisan prema ishodima

GenUI predstavlja fundamentalni pomak u pristupu dizajnu korisničkog interfejsa, prelazeći s fokusa na kreiranje specifičnih elemenata interfejsa na više holistički pristup orijentisan prema ishodima. Ovaj pomak ima nekoliko važnih implikacija:

1. **Fokus na ciljeve korisnika**:

    - Dizajneri će morati dublje razmišljati o ciljevima korisnika i željenim ishodima umjesto o specifičnim komponentama interfejsa.
    - Naglasak će biti na stvaranju sistema koji mogu generisati interfejse koji pomažu korisnicima da efikasno i efektivno postignu svoje ciljeve.
    - Pojaviće se novi UI okviri koji će AI-baziranim dizajnerima dati alate potrebne za generisanje korisničkih iskustava *u trenutku* i *iz početka* umjesto na osnovu unaprijed definisanih specifikacija ekrana.

2. **Promjena uloge dizajnera**:

    - Dizajneri će preći s kreiranja fiksnih layouta na definisanje pravila, ograničenja i smjernica koje AI sistemi trebaju slijediti pri generisanju interfejsa.
    - Moraće razviti vještine u područjima kao što su analiza podataka, inženjering AI promptova i sistemsko razmišljanje kako bi efikasno vodili GenUI sisteme.

3. **Važnost istraživanja korisnika**:

    - Istraživanje korisnika postaje još kritičnije u GenUI kontekstu, jer dizajneri trebaju razumjeti ne samo preferencije korisnika, već i kako se te preferencije i potrebe mijenjaju u različitim kontekstima.
    - Kontinuirano testiranje korisnika i petlje povratnih informacija bit će ključne za usavršavanje i poboljšanje sposobnosti AI-a da generiše efikasne interfejse.

4. **Dizajniranje za varijabilnost**:

    - Umjesto kreiranja jednog "savršenog" interfejsa, dizajneri će morati razmotriti višestruke moguće varijacije i osigurati da sistem može generisati odgovarajuće interfejse za različite potrebe korisnika.

- Ovo uključuje dizajniranje za granične slučajeve i osiguravanje da generisani interfejsi održavaju upotrebljivost i pristupačnost kroz različite konfiguracije.
- Diferencijacija proizvoda poprima nove dimenzije koje uključuju divergentne perspektive o psihologiji korisnika i leveraging jedinstvenih skupova podataka i baza znanja nedostupnih konkurentima.

Izazovi i razmatranja

Dok 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 generisanju istinski kontekstualno svjesnih interfejsa.
    - Problemi s performansama vezani za generisanje elemenata interfejsa u realnom vremenu, posebno na manje snažnim uređajima.
2. **Zahtjevi za podacima**:
    - Zavisno od slučaja upotrebe, efikasni GenUI sistemi mogu zahtijevati značajne količine korisničkih podataka za generisanje personalizovanih interfejsa.
    - 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 sa konstantno promjenjivim interfejsima mogla bi dovesti do problema upotrebljivosti, jer bi korisnici mogli imati poteškoća u pronalaženju poznatih elemenata ili efikasnoj navigaciji.
- Bit će ključno postići ravnotežu između personalizacije i održavanja konzistentnog, savladivog interfejsa.

4. **Pretjerano oslanjanje na AI**:

   - Postoji rizik od pretjeranog delegiranja dizajnerskih odluka AI sistemima, što potencijalno može dovesti do neinspirativnih, problematičnih ili jednostavno nefunkcionalnih izbora interfejsa.
   - Ljudski nadzor i mogućnost poništavanja AI-generisanih dizajna ostat će važni u doglednoj budućnosti.

5. **Problemi pristupačnosti**:

   - Osiguravanje da dinamički generisani interfejsi ostanu pristupačni korisnicima s invaliditetom predstavlja potpuno nove izazove, što je zabrinjavajuće s obzirom na loš nivo usklađenosti s pristupačnošću koji pokazuju tipični sistemi.
   - S druge strane, AI dizajneri mogu biti implementirani s *ugrađenom* brigom za pristupačnost i mogućnostima za izgradnju pristupačnih interfejsa u hodu, baš kao što grade UI za korisnike bez poteškoća.
   - U svakom slučaju, GenUI sistemi trebali bi biti dizajnirani s robusnim smjernicama za pristupačnost i procesima testiranja.

6. **Povjerenje korisnika i transparentnost**:

   - Korisnici se mogu osjećati neugodno s interfejsima koji "znaju previše" o njima ili se mijenjaju na načine koje ne razumiju.
   - Pružanje transparentnosti o tome kako i zašto se interfejsi personalizuju bit će važno za izgradnju povjerenja korisnika.

Budući izgledi i mogućnosti

Budućnost Generativnog UI-ja (GenUI) nosi ogromno obećanje za revoluciju načina na koji interagujemo s digitalnim proizvodima i uslugama. Kako se ova tehnologija nastavlja razvijati, možemo očekivati tektonski pomak u načinu na koji se korisnički interfejsi dizajniraju, implementiraju i doživljavaju. Mislim da je GenUI fenomen koji će konačno gurnuti naš softver u sferu onoga što se danas smatra naučnom fantastikom.

Jedna od najuzbudljivijih perspektiva GenUI-ja je njegov potencijal da unaprijedi pristupačnost u razmjeru koji nadilazi jednostavno osiguravanje da ljudi s ozbiljnim invaliditetom nisu potpuno isključeni iz korištenja vašeg softvera. Automatskim prilagođavanjem interfejsa individualnim potrebama korisnika, GenUI bi mogao učiniti digitalna iskustva inkluzivnijim nego ikad prije. Zamislite interfejse koji se besprijekorno prilagođavaju pružajući veći tekst za mlađe ili vizualno oštećene korisnike 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 dovesti do povećanog angažmana korisnika, zadovoljstva i lojalnosti kroz širok spektar digitalnih proizvoda. Kako interfejsi postaju više usklađeni s individualnim preferencijama i ponašanjima, korisnici će pronaći digitalna iskustva intuitivnijim i ugodnijim, što potencijalno vodi do dubljih i značajnijih interakcija s tehnologijom.

GenUI također ima potencijal transformisati proces uvođenja novih korisnika. Stvaranjem intuitivnih, personalizovanih iskustava za prve korisnike koja se brzo prilagođavaju nivou stručnosti svakog korisnika, GenUI bi mogao značajno smanjiti krivulju učenja povezanu s novim aplikacijama. To bi moglo dovesti do bržih stopa usvajanja i povećanog povjerenja korisnika u istraživanje novih funkcija i funkcionalnosti.

Još jedna uzbudljiva mogućnost je sposobnost GenUI-ja da održava konzistentno korisničko iskustvo na različitim uređajima i platformama, dok optimizira za svaki specifični kontekst upotrebe. Ovo bi moglo riješiti dugogodišnji izazov pružanja koherentnih

iskustava kroz sve fragmentiraniji pejzaž uređaja, od pametnih telefona i tableta do desktop računara i tehnologija u nastajanju poput naočala za proširenu stvarnost.

Priroda GenUI-ja zasnovana na podacima otvara mogućnosti za brzu iteraciju i poboljšanje u UI dizajnu. Prikupljanjem podataka u stvarnom vremenu o tome kako korisnici interaguju s generisanim interfejsima, dizajneri i programeri mogu steći neviđene uvide u ponašanje i preferencije korisnika. Ova povratna petlja mogla bi dovesti do kontinuiranih poboljšanja u UI dizajnu, vođenih stvarnim obrascima korištenja umjesto pretpostavkama ili ograničenim korisničkim testiranjem.

Da bi se pripremili za ovu promjenu, dizajneri će morati razvijati svoje vještine i način razmišljanja. Fokus će se pomjeriti sa stvaranja fiksnih rasporeda na razvoj sveobuhvatnih sistema dizajna i smjernica koje mogu informisati generisanje interfejsa vođeno AI-jem. Dizajneri će morati razviti duboko razumijevanje analize podataka, AI tehnologija i sistemskog razmišljanja kako bi efikasno vodili GenUI sisteme.

Štaviše, kako GenUI zamagljuje granice između dizajna i tehnologije, dizajneri će morati bliskije sarađivati s programerima i naučnicima koji se bave podacima. Ovaj interdisciplinarni pristup bit će ključan u stvaranju GenUI sistema koji nisu samo vizuelno privlačni i jednostavni za korištenje, već i tehnički robusni i etički ispravni.

Etičke implikacije GenUI-ja će također doći u prvi plan kako tehnologija sazrijeva. Dizajneri će igrati ključnu ulogu u razvoju okvira za odgovornu upotrebu vještačke inteligencije u dizajnu interfejsa, osiguravajući da personalizacija poboljšava korisničko iskustvo bez ugrožavanja privatnosti ili neetičke manipulacije ponašanjem korisnika.

Gledajući prema budućnosti, GenUI predstavlja i uzbudljive mogućnosti i značajne izazove. Ima potencijal da stvori intuitivnija, efikasnija i zadovoljavajuća digitalna iskustva za korisnike širom svijeta. Iako će zahtijevati od dizajnera da se prilagode i steknu nove vještine, također nudi neviđenu priliku za oblikovanje budućnosti interakcije između čovjeka i računara na dubok i smislen način. Put prema potpuno realizovanim GenUI sistemima će nesumnjivo biti složen, ali potencijalne nagrade u smislu poboljšanog korisničkog iskustva i digitalne pristupačnosti čine ga budućnošću za koju se vrijedi

zalagati.

Inteligentno Orkestriranje Radnih Tokova

U domenu razvoja aplikacija, radni tokovi igraju ključnu ulogu u definiranju kako se zadaci, procesi i interakcije s korisnicima strukturiraju i izvršavaju. Kako aplikacije postaju složenije, a očekivanja korisnika nastavljaju rasti, potreba za inteligentnim i adaptivnim orkestriranjem radnih tokova postaje sve očiglednija.

Pristup "Inteligentnog Orkestriranja Radnih Tokova" fokusira se na korištenje AI komponenti za dinamičko orkestriranje i optimizaciju složenih radnih tokova unutar aplikacija. Cilj je stvoriti aplikacije koje su efikasnije, responzivnije i prilagodljivije podacima i kontekstu u stvarnom vremenu.

U ovom poglavlju, istražit ćemo ključne principe i obrasce koji podupiru pristup inteligentnog orkestriranja radnih tokova. Razmotrit ćemo kako se AI može koristiti za

inteligentno usmjeravanje zadataka, automatizaciju donošenja odluka i dinamičko prilagođavanje radnih tokova na osnovu različitih faktora kao što su ponašanje korisnika, performanse sistema i poslovna pravila. Kroz praktične primjere i scenarije iz stvarnog svijeta, demonstrirat ćemo transformativni potencijal AI-ja u pojednostavljivanju i optimizaciji aplikacijskih radnih tokova.

Bez obzira gradite li poslovne aplikacije sa složenim poslovnim procesima ili aplikacije namijenjene potrošačima s dinamičkim korisničkim putovanjima, obrasci i tehnike o kojima se govori u ovom poglavlju opremit će vas znanjem i alatima za stvaranje inteligentnih i efikasnih radnih tokova koji poboljšavaju cjelokupno korisničko iskustvo i donose poslovnu vrijednost.

Poslovna Potreba

Tradicionalni pristupi upravljanju radnim tokovima često se oslanjaju na unaprijed definirane pravila i statička stabla odlučivanja, koja mogu biti kruta, nefleksibilna i nesposobna nositi se s dinamičnom prirodom modernih aplikacija.

Razmotrite scenarij gdje aplikacija za e-trgovinu treba upravljati složenim procesom ispunjenja narudžbe. Radni tok može uključivati više koraka kao što su validacija narudžbe, provjera zaliha, obrada plaćanja, dostava i obavještenja kupaca. Svaki korak može imati svoj set pravila, zavisnosti, vanjskih integracija i mehanizama za rukovanje izuzecima. Upravljanje takvim radnim tokom ručno ili kroz hardkodirano logiku može brzo postati glomazno, sklono greškama i teško za održavanje.

Štaviše, kako aplikacija skalira i broj istovremenih korisnika raste, radni tok se možda treba prilagoditi i optimizirati na osnovu podataka u stvarnom vremenu i performansi sistema. Na primjer, tokom perioda vršnog opterećenja, aplikacija možda treba dinamički prilagoditi radni tok kako bi prioritizirala određene zadatke, efikasno alocirala resurse i osigurala nesmetano korisničko iskustvo.

Tu nastupa pristup "Inteligentnog Orkestriranja Radnih Tokova". Korištenjem AI

komponenti, programeri mogu stvoriti radne tokove koji su inteligentni, adaptivni i samo-optimizirajući. AI može analizirati ogromne količine podataka, učiti iz prošlih iskustava i donositi informirane odluke u stvarnom vremenu za efikasno orkestriranje radnog toka.

Ključne Prednosti

1. **Povećana Efikasnost**: AI može optimizirati alokaciju zadataka, korištenje resursa i izvršavanje radnog toka, što vodi bržim vremenima obrade i poboljšanoj ukupnoj efikasnosti.
2. **Prilagodljivost**: Radni tokovi vođeni AI-jem mogu se dinamički prilagođavati promjenjivim uslovima, kao što su fluktuacije u korisničkoj potražnji, performanse sistema ili poslovni zahtjevi, osiguravajući da aplikacija ostane responsivna i otporna.
3. **Automatizirano Donošenje Odluka**: AI može automatizirati složene procese donošenja odluka unutar radnog toka, smanjujući manuelnu intervenciju i minimizirajući rizik ljudskih grešaka.
4. **Personalizacija**: AI može analizirati ponašanje korisnika, preference i kontekst kako bi personalizirao radni tok i isporučio prilagođena iskustva pojedinačnim korisnicima.
5. **Skalabilnost**: Radni tokovi pokretani AI-jem mogu se besprijekorno skalirati kako bi rukovali rastućim volumenima podataka i korisničkih interakcija, bez ugrožavanja performansi ili pouzdanosti.

U sljedećim sekcijama, istražit ćemo ključne obrasce i tehnike koje omogućavaju implementaciju inteligentnih radnih tokova i prikazati primjere iz stvarnog svijeta kako AI transformira upravljanje radnim tokovima u modernim aplikacijama.

Ključni Obrasci

Za implementaciju inteligentnog orkestriranja radnih tokova u aplikacijama, programeri mogu iskoristiti nekoliko ključnih obrazaca koji koriste snagu AI-ja. Ovi obrasci pružaju strukturirani pristup dizajniranju i upravljanju radnim tokovima, omogućavajući aplikacijama da se prilagode, optimiziraju i automatiziraju procese na osnovu podataka i konteksta u stvarnom vremenu. Hajde da istražimo neke od fundamentalnih obrazaca u inteligentnom orkestriranju radnih tokova.

Dinamičko Usmjeravanje Zadataka

Ovaj obrazac uključuje korištenje AI-ja za inteligentno usmjeravanje zadataka unutar radnog toka na osnovu različitih faktora kao što su prioritet zadatka, dostupnost resursa i performanse sistema. AI algoritmi mogu analizirati karakteristike svakog zadatka, razmotriti trenutno stanje sistema i donositi informirane odluke za dodjeljivanje zadataka najprikladnijim resursima ili putanjama obrade. Dinamičko usmjeravanje zadataka osigurava da su zadaci efikasno distribuirani i izvršeni, optimizirajući ukupne performanse radnog toka.

```
class TaskRouter
  include Raix::ChatCompletion
  include Raix::FunctionDispatch

  attr_accessor :task

  # list of functions that can be called by the AI entirely at its
  # discretion depending on the task received

  function :analyze_task_priority do
    TaskPriorityAnalyzer.perform(task)
  end

  function :check_resource_availability, # ...
  function :assess_system_performance, # ...
```

```
16    function :assign_task_to_resource, # ...
17
18    DIRECTIVE = "You are a task router, responsible for intelligently
19     assigning tasks to available resources based on priority, resource
20     availability, and system performance..."
21
22    def initialize(task)
23      self.task = task
24      transcript << { system: DIRECTIVE }
25      transcript << { user: task.to_json }
26    end
27
28    def perform
29      while task.unassigned?
30        chat_completion
31
32        # todo: add max loop counter and break
33      end
34
35      # capture the transcript for later analysis
36      task.update(routing_transcript: transcript)
37    end
38  end
```

Primijetite petlju kreiranu `while` izrazom u liniji 29, koja nastavlja slati upite AI-u sve dok zadatak nije dodijeljen. U liniji 35, čuvamo transkript zadatka za kasniju analizu i otklanjanje grešaka, ako to bude potrebno.

Kontekstualno odlučivanje

Možete koristiti vrlo sličan kod za donošenje kontekstualno svjesnih odluka unutar toka rada. Analiziranjem relevantnih podataka kao što su korisničke preference, historijski obrasci i unosi u realnom vremenu, AI komponente mogu odrediti najprikladniji tok akcije na svakoj tački odlučivanja u toku rada. Prilagodite ponašanje vašeg toka rada na osnovu specifičnog konteksta svakog korisnika ili scenarija, pružajući personalizirana i optimizirana iskustva.

Adaptivna kompozicija toka rada

Ovaj obrazac se fokusira na dinamičko komponovanje i prilagođavanje tokova rada na osnovu promjenjivih zahtjeva ili uslova. AI može analizirati trenutno stanje toka rada, identificirati uska grla ili neefikasnosti, i automatski modificirati strukturu toka rada kako bi optimizirao performanse. Adaptivna kompozicija toka rada omogućava aplikacijama da se kontinuirano razvijaju i poboljšavaju svoje procese bez potrebe za manuelnom intervencijom.

Rukovanje izuzecima i oporavak

Rukovanje izuzecima i oporavak su kritični aspekti inteligentne orkestracije toka rada. Kada radite sa AI komponentama i kompleksnim tokovima rada, neophodno je predvidjeti i elegantno rukovati izuzecima kako bi se osigurala stabilnost i pouzdanost sistema.

Evo nekoliko ključnih razmatranja i tehnika za rukovanje izuzecima i oporavak u inteligentnim tokovima rada:

1. **Propagacija izuzetaka**: Implementirajte konzistentan pristup za propagaciju izuzetaka kroz komponente toka rada. Kada se izuzetak pojavi unutar komponente, trebao bi biti uhvaćen, zabilježen i propagiran do orkestratora ili zasebne komponente odgovorne za rukovanje izuzecima. Ideja je centralizirati rukovanje izuzecima i spriječiti da izuzeci budu tiho progutani, kao i otvaranje mogućnosti za Inteligentno rukovanje greškama.
2. **Mehanizmi ponovnog pokušaja**: Mehanizmi ponovnog pokušaja pomažu u poboljšanju otpornosti toka rada i elegantnom rukovanju povremenim greškama. Svakako pokušajte implementirati mehanizme ponovnog pokušaja za prolazne ili oporavljive izuzetke, kao što su mrežna povezanost ili nedostupnost resursa koji se mogu automatski ponovno pokušati nakon određenog kašnjenja. Imati AI-pogonjen orkestrator ili rukovaoca izuzecima znači da vaše strategije ponovnog

pokušaja ne moraju biti mehaničke prirode, oslanjajući se na fiksne algoritme poput eksponencijalnog povlačenja. Možete prepustiti rukovanje ponovnim pokušajem "diskreciji" AI komponente odgovorne za odlučivanje kako rukovati izuzetkom.

3. **Rezervne strategije**: Ako AI komponenta ne uspije pružiti validan odgovor ili naiđe na grešku—česta pojava s obzirom na njenu naprednu prirodu—imajte rezervni mehanizam koji će osigurati da se tok rada može nastaviti. Ovo može uključivati korištenje zadanih vrijednosti, alternativnih algoritama, ili Čovjeka u petlji za donošenje odluka i održavanje toka rada u pokretu.
4. **Kompenzacijske akcije**: Direktive orkestratora bi trebale uključivati instrukcije o kompenzacijskim akcijama za rukovanje izuzecima koji se ne mogu automatski riješiti. Kompenzacijske akcije su koraci poduzeti za poništavanje ili ublažavanje efekata neuspjele operacije. Na primjer, ako korak obrade plaćanja ne uspije, kompenzacijska akcija bi mogla biti povrat transakcije i obavještavanje korisnika. Kompenzacijske akcije pomažu u održavanju konzistentnosti podataka i integriteta u slučaju izuzetaka.
5. **Praćenje i upozoravanje o izuzecima**: Postavite mehanizme za praćenje i upozoravanje kako biste detektirali i obavijestili relevantne sudionike o kritičnim izuzecima. Orkestrator može biti svjestan pragova i pravila za aktiviranje upozorenja kada izuzeci pređu određene granice ili kada se pojave specifični tipovi izuzetaka. Ovo omogućava proaktivnu identifikaciju i rješavanje problema prije nego što utječu na cjelokupni sistem.

Evo primjera rukovanja izuzecima i oporavka u Ruby komponenti toka 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, `InventoryManager` komponenta provjerava dostupnost proizvoda za datu narudžbu. Ako je dostupna količina nedovoljna, podiže `InsufficientInventoryError`. Izuzetak se hvata, bilježi, i implementira se mehanizam ponovnog pokušaja. Ako se prekorači ograničenje ponovnih pokušaja, komponenta prelazi na ručnu intervenciju obavještavanjem administratora.

Implementiranjem robusnog rukovanja izuzecima i mehanizama oporavka, možete osigurati da vaši inteligentni radni tokovi budu otporni, održivi i sposobni da elegantno

rukuju neočekivanim situacijama.

Ovi obrasci čine temelj orkestracije inteligentnog radnog toka i mogu se kombinovati i prilagoditi specifičnim zahtjevima različitih aplikacija. Koristeći ove obrasce, programeri mogu kreirati radne tokove koji su fleksibilni, otporni i optimizovani 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 da ilustrujemo integraciju AI komponenti u upravljanje radnim tokom.

Implementacija Orkestracije Inteligentnog Radnog Toka u Praksi

Sada kada smo istražili ključne obrasce u orkestraciji inteligentnog radnog toka, hajde da se udubimo u to kako se ovi obrasci mogu implementirati u aplikacijama iz stvarnog svijeta. Pružit ćemo praktične primjere i isječke koda da ilustrujemo integraciju AI komponenti u upravljanje radnim tokom.

Inteligentni Procesor Narudžbi

Hajde da se udubimo u praktičan primjer implementacije orkestracije inteligentnog radnog toka koristeći AI-pogonjen `OrderProcessor` komponentu u Ruby on Rails e-commerce aplikaciji. `OrderProcessor` realizuje koncept Upravitelja Procesa Integracije u Preduzeću koji smo prvi put sreli u Poglavlju 3 kada smo diskutovali o Mnoštvu Radnika. Komponenta će biti odgovorna za upravljanje tokom ispunjavanja narudžbe, donošenje odluka o usmjeravanju na osnovu međurezultata i orkestraciju izvršenja različitih koraka obrade.

Proces ispunjavanja narudžbe uključuje više koraka kao što su validacija narudžbe, provjera inventara, obrada plaćanja i isporuka. Svaki korak je implementiran kao zaseban radni proces koji obavlja specifičan zadatak i vraća rezultat `OrderProcessor`-u. Koraci nisu obavezni i ne moraju se nužno izvršavati određenim redoslijedom.

Evo primjera implementacije `OrderProcessor`-a. Sadrži dva mixin-a iz Raix. Prvi (`ChatCompletion`) daje mu mogućnost chat završetka, što ga čini AI komponentom. Drugi (`FunctionDispatch`) omogućava pozivanje funkcija od strane AI-a, dozvoljavajući mu da odgovori na prompt pozivom funkcije umjesto tekstualne poruke.

Radne funkcije (`validate_order`, `check_inventory`, i ostale) delegiraju svojim odgovarajućim radnim klasama, koje mogu biti AI ili ne-AI komponente, sa jedinim zahtjevom da vrate rezultate svog rada u formatu koji se može predstaviti kao string.

Kao i sa svim drugim primjerima u ovom dijelu knjige, ovaj kod je praktično pseudo-kod i namijenjen je samo da prenese značenje obrasca i inspiriše vaše vlastite kreacije. Potpuni opisi obrazaca i kompletni primjeri koda su uključeni u Dijelu 2.

```
class OrderProcessor
  include Raix::ChatCompletion
  include Raix::FunctionDispatch

  SYSTEM_DIRECTIVE = "You are an order processor, tasked with..."

  def initialize(order)
    self.order = order
    transcript << { system: SYSTEM_DIRECTIVE }
    transcript << { user: order.to_json }
  end

  def perform
    # will continue looping until `stop_looping!` is called
    chat_completion(loop: true)
  end

```

```
  # list of functions available to be called by the AI
  # truncated for brevity

  def functions
    [
      {
        name: "validate_order",
        description: "Invoke to check validity of order",
        parameters: {
          ...
      },
      ...
    ]
  end

  # implementation of functions that can be called by the AI
  # entirely at its discretion, depending on the needs of the order

  def validate_order
    OrderValidationWorker.perform(@order)
  end

  def check_inventory
    InventoryCheckWorker.perform(@order)
  end

  def process_payment
    PaymentProcessingWorker.perform(@order)
  end

  def schedule_shipping
    ShippingSchedulerWorker.perform(@order)
  end

  def send_confirmation
    OrderConfirmationWorker.perform(@order)
  end

  def finished_processing
    @order.update!(transcript:, processed_at: Time.current)
    stop_looping!
  end
```

```
60 end
```

U primjeru, OrderProcessor je inicijaliziran s objektom narudžbe i održava transkript izvršenja toka rada, u tipičnom formatu transkripta konverzacije koji je svojstven velikim jezičkim modelima. AI-ju je data potpuna kontrola nad orkestracijom izvršenja različitih koraka obrade, kao što su validacija narudžbe, provjera zaliha, obrada plaćanja i isporuka.

Svaki put kada se pozove metoda `chat_completion`, transkript se šalje AI-ju da 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 nizak nivo zaliha, `OrderProcessor` može zakazati zadatak dopune. Ako obrada plaćanja ne uspije, može pokrenuti ponovni pokušaj ili obavijestiti korisničku podršku.

> Gore navedeni primjer nema definisane funkcije za dopunu ili obavještavanje korisničke podrške, ali apsolutno bi mogao imati.

Transkript raste svaki put kada se pozove funkcija i služi kao zapis izvršenja toka rada, uključujući rezultate svakog koraka i AI-generisana uputstva za sljedeće korake. Ovaj transkript se može koristiti za debugiranje, reviziju i pružanje uvida u proces ispunjenja narudžbe.

Korištenjem AI-ja u `OrderProcessor`-u, e-commerce aplikacija može dinamički prilagoditi tok rada na osnovu podataka u realnom vremenu i inteligentno upravljati izuzecima. AI komponenta može donositi informisane odluke, optimizirati tok rada i osigurati nesmetanu obradu narudžbi čak i u složenim scenarijima.

Činjenica da je jedini zahtjev za radne procese da vrate neki razumljiv izlaz koji AI može razmotriti pri odlučivanju šta dalje učiniti, možda će vam početi ukazivati na to kako ovaj pristup može smanjiti posao mapiranja ulaza/izlaza koji je obično uključen

pri integraciji različitih sistema međusobno.

Inteligentni moderator sadržaja

Aplikacije društvenih medija općenito zahtijevaju barem minimalnu moderaciju sadržaja kako bi se osigurala sigurna i zdrava zajednica. Ovaj primjer komponente `ContentModerator` koristi AI za inteligentnu orkestraciju toka moderacije, donoseći odluke na osnovu karakteristika sadržaja i rezultata različitih koraka moderacije.

Proces moderacije uključuje više koraka kao što su analiza teksta, prepoznavanje slika, procjena reputacije korisnika i ručni pregled. Svaki korak je implementiran kao zaseban radni proces koji obavlja specifičan 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

```

```
  # list of functions available to be called by the AI
  # truncated for brevity

  def functions
    [
      {
        name: "analyze_text",
        # ...
      },
      {
        name: "recognize_image",
        description: "Invoke to describe images...",
        # ...
      },
      {
        name: "assess_user_reputation",
        # ...
      },
      {
        name: "escalate_to_manual_review",
        # ...
      },
      {
        name: "approve_content",
        # ...
      },
      {
        name: "reject_content",
        # ...
      }
    ]
  end

  # implementation of functions that can be called by the AI
  # entirely at its discretion, depending on the needs of the order

  def analyze_text
    result = TextAnalysisWorker.perform(@content)
    continue_with(result)
  end

  def recognize_image
```

```
    result = ImageRecognitionWorker.perform(@content)
    continue_with(result)
  end

  def assess_user_reputation
    result = UserReputationWorker.perform(@content.user)
    continue_with(result)
  end

  def escalate_to_manual_review
    ManualReviewWorker.perform(@content)
    @content.update!(status: 'pending', transcript: @transcript)
  end

  def approve_content
    @content.update!(status: 'approved', transcript: @transcript)
  end

  def reject_content
    @content.update!(status: 'rejected', transcript: @transcript)
  end

  private

  def continue_with(result)
    @transcript << { function: result }
    complete(@transcript)
  end
end
```

U ovom primjeru, `ContentModerator` je inicijaliziran sa objektom sadržaja i održava zapisnik moderacije u konverzacijskom formatu. AI komponenta ima potpunu kontrolu nad tokom moderacije, odlučujući koje korake izvršiti na osnovu 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 zapisnik moderacije, sa izuzetkom

funkcije za eskalaciju, 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 vizuelnom pregledu. Ako bilo koji radni proces upozori na potencijalno štetan sadržaj, AI može odlučiti da eskalira sadržaj na ručni pregled ili ga direktno odbiti. Ali ovisno o ozbiljnosti upozorenja, AI može odlučiti da koristi rezultate procjene reputacije korisnika pri odlučivanju kako postupiti sa sadržajem u kojem nije siguran. Ovisno o slučaju upotrebe, možda pouzdani korisnici imaju više slobode u onome što mogu objaviti. I tako dalje...

Kao i u prethodnom primjeru upravitelja procesa, zapisnik moderacije služi kao evidencija izvršenja toka rada, uključujući rezultate svakog koraka i odluke koje je generirao AI. Ovaj zapisnik se može koristiti za reviziju, transparentnost i unapređenje procesa moderacije tokom vremena.

Korištenjem AI-ja u `ContentModerator`-u, aplikacija društvenih medija može dinamički prilagoditi tok moderacije na osnovu karakteristika sadržaja i inteligentno upravljati složenim scenarijima moderacije. AI komponenta može donositi informirane odluke, optimizirati tok rada i osigurati sigurno i zdravo iskustvo zajednice.

Istražimo još dva primjera koji demonstriraju prediktivno raspoređivanje zadataka i rukovanje izuzecima i oporavak u kontekstu inteligentnog orkestriranja toka rada.

Prediktivno Raspoređivanje Zadataka u Sistemu Korisničke Podrške

U aplikaciji za korisničku podršku izrađenoj pomoću Ruby on Rails, efikasno upravljanje i prioritiziranje tiketa za podršku je ključno za pružanje pravovremene pomoći korisnicima. Komponenta `SupportTicketScheduler` koristi AI za prediktivno raspoređivanje i dodjeljivanje tiketa za podršku dostupnim agentima na osnovu različitih faktora kao što su hitnost tiketa, stručnost agenta i radno opterećenje.

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

  SYSTEM_DIRECTIVE = "You are a support ticket scheduler,
    tasked with intelligently assigning tickets to available agents..."

  def initialize(ticket)
    @ticket = ticket
    @transcript = [
      { system: SYSTEM_DIRECTIVE },
      { user: ticket.to_json }
    ]
  end

  def perform
    complete(@transcript)
  end

  def model
    "openai/gpt-4"
  end

  def functions
    [
      {
        name: "analyze_ticket_urgency",
        # ...
      },
      {
        name: "list_available_agents",
        description: "Includes expertise of available agents",
        # ...
      },
      {
        name: "predict_agent_workload",
        description: "Uses historical data to predict upcoming workloads",
        # ...
      },
      {
        name: "assign_ticket_to_agent",
        # ...
```

```
      },
      {
        name: "reschedule_ticket",
        # ...
      }
    ]
  end

  # implementation of functions that can be called by the AI
  # entirely at its discretion, depending on the needs of the order

  def analyze_ticket_urgency
    result = TicketUrgencyAnalyzer.perform(@ticket)
    continue_with(result)
  end

  def list_available_agents
    result = ListAvailableAgents.perform
    continue_with(result)
  end

  def predict_agent_workload
    result = AgentWorkloadPredictor.perform
    continue_with(result)
  end

  def assign_ticket_to_agent
    TicketAssigner.perform(@ticket, @transcript)
  end

  def delay_assignment(until)
    until = DateTimeStandardizer.process(until)
    SupportTicketScheduler.delay(@ticket, @transcript, until)
  end

  private

  def continue_with(result)
    @transcript << { function: result }
    complete(@transcript)
  end
end
```

U ovom primjeru, `SupportTicketScheduler` je inicijaliziran s objektom tiketa za podršku i održava zapisnik rasporeda. AI komponenta analizira detalje tiketa i prediktivno planira dodjelu tiketa na osnovu faktora kao što su hitnost tiketa, stručnost agenta i predviđeno radno opterećenje 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 i dodaje rezultat u zapisnik rasporeda. AI također ima opciju odgoditi dodjelu koristeći funkciju `delay_assignment`.

AI komponenta pregledava zapisnik rasporeda 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 planiranja zadataka, aplikacija za korisničku podršku može optimizirati dodjelu tiketa, smanjiti vrijeme odziva i poboljšati ukupno zadovoljstvo korisnika. Proaktivno i efikasno upravljanje tiketima za podršku osigurava da pravi tiketi budu dodijeljeni pravim agentima u pravo vrijeme.

Upravljanje izuzecima i oporavak u procesu obrade podataka

Upravljanje izuzecima i oporavak od grešaka je ključno za osiguranje integriteta podataka i sprječavanje gubitka podataka. Komponenta `DataProcessingOrchestrator` koristi AI za inteligentno upravljanje izuzecima i orkestraciju procesa oporavka u procesu obrade podataka.

```
class DataProcessingOrchestrator
  include Raix::ChatCompletion
  include Raix::FunctionDispatch

  SYSTEM_DIRECTIVE = "You are a data processing orchestrator..."

  def initialize(data_batch)
    @data_batch = data_batch
    @transcript = [
      { system: SYSTEM_DIRECTIVE },
      { user: data_batch.to_json }
    ]
  end

  def perform
    complete(@transcript)
  end

  def model
    "openai/gpt-4"
  end

  def functions
    [
      {
        name: "validate_data",
        # ...
      },
      {
        name: "process_data",
        # ...
      },
      {
        name: "request_fix",
        # ...
      },
      {
        name: "retry_processing",
        # ...
      },
      {
        name: "mark_data_as_failed",
```

```
        # ...
      },
      {
        name: "finished",
        # ...
      }
    ]
  end

  # implementation of functions that can be called by the AI
  # entirely at its discretion, depending on the needs of the order

  def validate_data
    result = DataValidator.perform(@data_batch)
    continue_with(result)
  rescue ValidationException => e
    handle_validation_exception(e)
  end

  def process_data
    result = DataProcessor.perform(@data_batch)
    continue_with(result)
  rescue ProcessingException => e
    handle_processing_exception(e)
  end

  def request_fix(description_of_fix)
    result = SmartDataFixer.new(description_of_fix, @data_batch)
    continue_with(result)
  end

  def retry_processing(timeout_in_seconds)
    wait(timeout_in_seconds)
    process_data
  end

  def mark_data_as_failed
    @data_batch.update!(status: 'failed', transcript: @transcript)
  end

  def finished
    @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 orkestira tok obrade podataka, upravljajući izuzecima i oporavljajući se od grešaka po potrebi.

Dostupne funkcije koje AI može pozvati uključuju `validate_data`, `process_data`, `request_fix`, `retry_processing`, i `mark_data_as_failed`. Svaka funkcija delegira zadatak odgovarajućoj komponenti za obradu podataka i dodaje rezultat ili detalje izuzetka u transkript obrade.

Ako se pojavi izuzetak validacije tokom koraka `validate_data`, funkcija `handle_validation_exception` dodaje podatke o izuzetku u transkript i vraća kontrolu AI-u. Slično tome, ako se pojavi izuzetak obrade tokom koraka `process_data`, AI može odlučiti o strategiji oporavka.

Ovisno o prirodi susretnutog izuzetka, AI može po vlastitom nahođenju odlučiti da pozove `request_fix`, koji delegira AI-pokrenutoj komponenti `SmartDataFixer` (pogledajte poglavlje o Samozacjeljujućim Podacima). Program za popravak podataka

dobija običan opis na engleskom jeziku o tome kako treba modificirati `@data_batch` tako da se obrada može ponovo pokušati. Možda bi uspješno ponovno pokušavanje podrazumijevalo uklanjanje zapisa iz grupe podataka koji nisu prošli validaciju i/ili njihovo kopiranje u drugi tok obrade za ljudski pregled? Mogućnosti su gotovo beskrajne.

Uključivanjem AI-vođenog upravljanja izuzecima i oporavka, aplikacija za obradu podataka postaje otpornija i tolerantnija na greške. `DataProcessingOrchestrator` inteligentno upravlja izuzecima, minimizira gubitak podataka i osigurava neometano izvršavanje toka obrade podataka.

Praćenje i Bilježenje

Praćenje i bilježenje pružaju uvid u napredak, performanse i zdravlje komponenti toka rada pokrenutih AI-em, omogućavajući programerima da prate i analiziraju ponašanje sistema. Implementacija efektivnih mehanizama za praćenje i bilježenje je ključna za otklanjanje grešaka, reviziju i kontinuirano poboljšanje inteligentnih tokova rada.

Praćenje Napretka i Performansi Toka Rada

Da bi se osiguralo neometano izvršavanje inteligentnih tokova rada, važno je pratiti napredak i performanse svake komponente toka rada. Ovo uključuje praćenje ključnih metrika i događaja tokom životnog ciklusa toka rada.

Neki važni aspekti za praćenje uključuju:

1. **Vrijeme Izvršavanja Toka Rada**: Mjerenje vremena koje je potrebno svakoj komponenti toka rada da završi svoj zadatak. Ovo pomaže u identifikaciji uskih grla u performansama i optimizaciji ukupne efikasnosti toka rada.

2. **Iskorištenost Resursa**: Praćenje iskorištenosti sistemskih resursa, kao što su CPU, memorija i skladište, od strane svake komponente toka rada. Ovo pomaže osigurati da sistem radi unutar svojih kapaciteta i može efektivno upravljati radnim opterećenjem.

3. **Stope Grešaka i Izuzeci**: Praćenje pojave grešaka i izuzetaka unutar komponenti toka rada. Ovo pomaže u identifikaciji potencijalnih problema i omogućava proaktivno upravljanje greškama i oporavak.

4. **Tačke Odlučivanja i Ishodi**: Praćenje tačaka odlučivanja unutar toka rada i ishoda odluka donesenih od strane AI-a. Ovo pruža uvid u ponašanje i efektivnost AI komponenti.

Podaci prikupljeni procesom praćenja mogu biti prikazani na kontrolnim pločama ili korišteni kao ulazni podaci za planirane izvještaje koji informišu administratore sistema o zdravlju sistema.

Podaci praćenja mogu se proslijediti AI-pokrenutom procesu sistemskog administratora 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 izuzecima koji se javljaju tokom izvršavanja toka rada.

Neki važni aspekti za bilježenje uključuju:

1. **Pokretanje i Završetak Toka Rada**: Bilježenje vremena početka i završetka svake instance toka rada, zajedno sa svim relevantnim metapodacima kao što su ulazni podaci i korisnički kontekst.

2. **Izvršavanje Komponenti**: Bilježenje detalja izvršavanja svake komponente toka rada, uključujući ulazne parametre, izlazne rezultate i sve generirane međupodatke.

3. **AI Odluke i Obrazloženja**: Bilježenje odluka donesenih od strane AI komponenti, zajedno s pripadajućim obrazloženjem ili ocjenama pouzdanosti. Ovo pruža transparentnost i omogućava reviziju odluka donesenih od strane AI-a.

4. **Izuzeci i Poruke o Greškama**: Bilježenje svih izuzetaka ili poruka o greškama na koje se naišlo tokom izvršavanja toka rada, uključujući stack trace i relevantne kontekstualne informacije.

Bilježenje se može implementirati koristeći različite tehnike, kao što su pisanje u log datoteke, pohranjivanje logova u bazu podataka ili slanje logova centraliziranom servisu za bilježenje. Važno je odabrati framework za bilježenje koji pruža fleksibilnost, skalabilnost i jednostavnu integraciju s arhitekturom aplikacije.

Evo primjera kako se bilježenje može implementirati u Ruby on Rails aplikaciji koristeći klasu `ActiveSupport::Logger`:

```
class WorkflowLogger
  def self.log(message, severity = :info)
    @logger ||= ActiveSupport::Logger.new('workflow.log')
    @logger.formatter ||= proc do |severity, datetime, progname, msg|
      "#{datetime} [#{severity}] #{msg}\n"
    end
    @logger.send(severity, message)
  end
end

# Usage example
WorkflowLogger.log("Workflow initiated for order ##{@order.id}")
WorkflowLogger.log("Payment processing completed successfully")
WorkflowLogger.log("Inventory check failed for item ##{item.id}", :error)
```

Strateškim postavljanjem izjava za bilježenje kroz komponente radnog toka i tačke odlučivanja umjetne inteligencije, programeri mogu prikupiti vrijedne informacije za otklanjanje grešaka, reviziju i analizu.

Prednosti Praćenja i Bilježenja

Implementacija praćenja i bilježenja u inteligentnoj orkestraciji radnih tokova nudi nekoliko prednosti:

1. **Otklanjanje Grešaka i Rješavanje Problema**: Detaljni zapisi i podaci praćenja pomažu programerima da brzo identificiraju i dijagnosticiraju probleme. Oni pružaju uvid u tok izvršavanja radnog toka, interakcije komponenti i sve greške ili izuzetke na koje se naiđe.

2. **Optimizacija Performansi**: Praćenje metrika performansi omogućava programerima da identificiraju uska grla i optimiziraju komponente radnog toka za bolju efikasnost. Analizirajući vremena izvršavanja, korištenje resursa i druge metrike, programeri mogu donositi informirane odluke za poboljšanje ukupnih performansi sistema.

3. **Revizija i Usklađenost**: Bilježenje ključnih događaja i odluka pruža revizorski trag za regulatornu usklađenost i odgovornost. To omogućava organizacijama da prate i verificiraju akcije koje poduzimaju AI komponente i osiguraju pridržavanje poslovnih pravila i zakonskih zahtjeva.

4. **Kontinuirano Poboljšanje**: Podaci praćenja i bilježenja služe kao vrijedni ulazni podaci za kontinuirano poboljšanje inteligentnih radnih tokova. Analizom historijskih podataka, identifikacijom obrazaca i mjerenjem efektivnosti AI odluka, programeri mogu iterativno usavršavati i unapređivati logiku orkestracije radnih tokova.

Razmatranja i Najbolje Prakse

Pri implementaciji praćenja i bilježenja u inteligentnoj orkestraciji radnih tokova, razmotrite sljedeće najbolje prakse:

1. **Definiranje Jasnih Metrika Praćenja**: Identificirajte ključne metrike i događaje koje treba pratiti na osnovu specifičnih zahtjeva radnog toka. Fokusirajte se na metrike koje pružaju smislene uvide u performanse, zdravlje i ponašanje sistema.

2. **Implementacija Granularnog Bilježenja**: Osigurajte da su izjave za bilježenje postavljene na odgovarajućim tačkama unutar komponenti radnog toka i tačkama odlučivanja AI-a. Zabilježite relevantne kontekstualne informacije, kao što su ulazni parametri, izlazni rezultati i svi međupodaci koji se generiraju.

3. **Korištenje Strukturiranog Bilježenja**: Usvojite format strukturiranog bilježenja kako biste olakšali jednostavno parsiranje i analizu podataka zapisa. Strukturirano bilježenje omogućava bolju pretraživost, 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 odgovarajući period zadržavanja na osnovu 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, kao što su lični identifikacijski podaci (PII) ili povjerljivi poslovni podaci. Implementirajte odgovarajuće sigurnosne mjere, kao što su maskiranje podataka ili enkripcija, kako biste zaštitili osjetljive informacije u datotekama zapisa.

6. **Integracija sa Alatima za Praćenje i Upozoravanje**: Iskoristite alate za praćenje i upozoravanje za centralizaciju prikupljanja, analize i vizualizacije podataka praćenja i bilježenja. Ovi alati mogu pružiti uvide u realnom vremenu, generirati upozorenja na osnovu predefiniranih pragova i olakšati proaktivno otkrivanje i rješavanje problema. Moj omiljeni od ovih alata je Datadog.

Implementacijom sveobuhvatnih mehanizama praćenja i bilježenja, programeri mogu steći vrijedne uvide u ponašanje i performanse inteligentnih radnih tokova. Ovi uvidi omogućavaju efikasno otklanjanje grešaka, optimizaciju i kontinuirano poboljšanje sistema za orkestraciju radnih tokova zasnovanih na AI-u.

Razmatranja o Skalabilnosti i Performansama

Skalabilnost i performanse su kritični aspekti koje treba razmotriti pri dizajniranju i implementaciji sistema za inteligentnu orkestraciju radnih tokova. Kako se povećava obim konkurentnih radnih tokova i složenost komponenti zasnovanih na AI-u, postaje

ključno osigurati da sistem može efikasno upravljati radnim opterećenjem i besprijekorno se skalirati kako bi zadovoljio rastuće zahtjeve.

Rukovanje Velikim Obimom Konkurentnih Radnih Tokova

Sistemi za inteligentnu orkestraciju radnih tokova često moraju rukovati velikim brojem konkurentnih radnih tokova. Da biste osigurali skalabilnost, razmotrite sljedeće strategije:

1. **Asinhrona Obrada**: Implementirajte mehanizme asinhrone obrade za odvajanje izvršavanja komponenti radnog toka. Ovo omogućava sistemu da upravlja višestrukim radnim tokovima istovremeno bez blokiranja ili čekanja da svaka komponenta završi. Asinhrona obrada se može postići korištenjem redova poruka, arhitektura vođenih događajima ili okvira za obradu pozadinskih poslova kao što je Sidekiq.

2. **Distribuirana Arhitektura**: Dizajnirajte arhitekturu sistema da koristi serverless komponente (kao što je AWS Lambda) ili jednostavno distribuirajte radno opterećenje preko više čvorova ili servera zajedno sa vašim glavnim aplikacijskim serverom. Ovo omogućava horizontalnu skalabilnost, gdje se dodatni čvorovi mogu dodati za rukovanje povećanim obimom radnih tokova.

3. **Paralelno Izvršavanje**: Identificirajte mogućnosti za paralelno izvršavanje unutar radnih tokova. Neke komponente radnog toka mogu biti nezavisne jedna od druge i mogu se izvršavati istovremeno. Korištenjem tehnika paralelne obrade, kao što su višenitnost ili distribuirani redovi zadataka, sistem može optimizirati korištenje resursa i smanjiti ukupno vrijeme izvršavanja radnog toka.

Optimizacija Performansi AI-Pogođenih Komponenti

AI-pogođene komponente, kao što su modeli mašinskog učenja ili sistemi za obradu prirodnog jezika, mogu biti računski zahtjevne i uticati na ukupne performanse sistema

za orkestraciju radnih tokova. Da biste optimizirali performanse AI komponenti, razmotrite sljedeće tehnike:

1. **Keširanje**: Ako je vaša AI obrada čisto generativna i ne uključuje pretraživanje informacija u realnom vremenu ili eksterne integracije za generisanje chat odgovora, možete istražiti mehanizme keširanja za pohranu i ponovno korištenje rezultata često pristupanih ili računski zahtjevnih operacija.

2. **Optimizacija Modela**: Kontinuirano optimizirajte način na koji koristite AI modele u komponentama radnog toka. Ovo može uključivati tehnike kao što je *Destilacija Promptova* ili jednostavno testiranje novih modela kako postaju dostupni.

3. **Grupna Obrada**: Ako radite sa modelima klase GPT-4, možda ćete moći iskoristiti tehnike grupne obrade za obradu više podatkovnih tačaka ili zahtjeva u jednoj grupi, umjesto njihove pojedinačne obrade. Obradom podataka u grupama, sistem može optimizirati korištenje resursa i smanjiti opterećenje ponovljenih zahtjeva modela.

Praćenje i Profiliranje Performansi

Za identifikaciju uskih grla u performansama i optimizaciju skalabilnosti inteligentnog sistema za orkestraciju radnih tokova, ključno je implementirati mehanizme za praćenje i profiliranje. Razmotrite sljedeće pristupe:

1. **Metrike Performansi**: Definirajte i pratite ključne metrike performansi, kao što su vrijeme odziva, propusnost, iskorištenost resursa i latencija. Ove metrike pružaju uvid u performanse sistema i pomažu u identifikaciji područja za optimizaciju. Popularni AI model agregator OpenRouter uključuje metrike Hosta[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 AI operacija. Alati za profiliranje mogu pomoći

[1] Host je vrijeme potrebno za primanje prvog bajta streamovanog generisanja od strane model hosta, poznatog kao "vrijeme do prvog bajta."

[2] Brzina se izračunava kao broj tokena završetka podijeljenih s ukupnim vremenom generisanja. Za nestreamovane zahtjeve, latencija se smatra dijelom vremena generisanja.

u identifikaciji kritičnih tačaka performansi, neefikasnih putanja koda ili operacija koje intenzivno koriste resurse. Popularni alati za profiliranje uključuju New Relic, Scout, ili ugrađene profilere koje pruža programski jezik ili framework.

3. **Testiranje Opterećenja**: Provodite testiranje opterećenja kako biste procijenili performanse sistema pod različitim nivoima istovremenih radnih opterećenja. Testiranje opterećenja pomaže u identifikaciji granica skalabilnosti sistema, otkrivanju degradacije performansi i osiguravanju da sistem 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 kontrolne table za praćenje i upozorenja za praćenje ključnih indikatora performansi (KPI) i primanje obavještenja kada se prekorače unaprijed definirani pragovi. Ovo omogućava brzu identifikaciju i rješavanje problema s performansama.

Strategije Skaliranja

Za upravljanje povećanim radnim opterećenjima i osiguravanje skalabilnosti inteligentnog sistema za orkestraciju radnih tokova, razmotrite sljedeće strategije skaliranja:

1. **Vertikalno Skaliranje**: Vertikalno skaliranje uključuje povećanje resursa (npr. CPU, memorija) pojedinačnih čvorova ili servera za upravljanje većim radnim opterećenjima. Ovaj pristup je pogodan kada sistem zahtijeva više procesorske snage ili memorije za upravljanje složenim radnim tokovima ili AI operacijama.

2. **Horizontalno Skaliranje**: Horizontalno skaliranje uključuje dodavanje više čvorova ili servera sistemu za distribuciju radnog opterećenja. Ovaj pristup je efikasan kada sistem treba upravljati velikim brojem istovremenih radnih tokova ili kada se radno opterećenje može lako distribuirati preko više čvorova. Horizontalno skaliranje zahtijeva distribuiranu arhitekturu i mehanizme balansiranja opterećenja za osiguranje ravnomjerne distribucije prometa.

3. **Automatsko Skaliranje**: Implementirajte mehanizme automatskog skaliranja za automatsko prilagođavanje broja čvorova ili resursa na osnovu zahtjeva radnog opterećenja. Automatsko skaliranje omogućava sistemu da se dinamički skalira gore ili dolje u zavisnosti od dolaznog prometa, osiguravajući optimalnu iskorištenost resursa i ekonomičnost. Cloud platforme poput Amazon Web Services (AWS) ili Google Cloud Platform (GCP) pružaju mogućnosti automatskog skaliranja koje se mogu iskoristiti za inteligentne sisteme orkestracije radnih tokova.

Tehnike Optimizacije Performansi

Pored strategija skaliranja, razmotrite sljedeće tehnike optimizacije performansi za poboljšanje efikasnosti inteligentnog sistema za orkestraciju radnih tokova:

1. **Efikasno Skladištenje i Preuzimanje Podataka**: Optimizirajte mehanizme skladištenja i preuzimanja podataka koje koriste komponente radnog toka. Koristite efikasno indeksiranje baze podataka, tehnike optimizacije upita i keširanje podataka kako biste minimizirali latenciju i poboljšali performanse operacija intenzivnih podataka.

2. **Asinhroni U/I**: Koristite asinhrone U/I operacije kako biste spriječili blokiranje i poboljšali odziv sistema. Asinhroni U/I omogućava sistemu da istovremeno obrađuje više zahtjeva bez čekanja na završetak U/I operacija, čime se maksimizira iskorištenost resursa.

3. **Efikasna serijalizacija i deserijalizacija**: Optimizirajte procese serijalizacije i deserijalizacije koji se koriste za razmjenu podataka između komponenti toka rada. Koristite efikasne formate serijalizacije, kao što su Protocol Buffers ili MessagePack, kako biste smanjili opterećenje serijalizacije podataka i poboljšali performanse komunikacije između komponenti.

Za aplikacije zasnovane na Ruby-ju, razmotrite korištenje Universal ID. Universal ID koristi i MessagePack i Brotli (kombinaciju napravljenu za brzinu i najbolju kompresiju podataka u svojoj klasi). Kada se kombinuju, ove biblioteke su do 30% brže i imaju stope kompresije unutar 2-5% u poređenju sa Protocol Buffers.

4. **Kompresija i kodiranje**: Primijenite tehnike kompresije i kodiranja kako biste smanjili veličinu podataka koji se prenose između komponenti toka rada. Algoritmi za kompresiju, kao što su gzip ili Brotli, mogu značajno smanjiti korištenje mrežnog propusnog opsega i poboljšati ukupne performanse sistema.

Uzimajući u obzir aspekte skalabilnosti i performansi tokom dizajna i implementacije sistema za inteligentnu orkestraciju toka rada, možete osigurati da vaš sistem može obraditi velike količine konkurentnih tokova rada, optimizirati performanse komponenti zasnovanih na vještačkoj inteligenciji i nesmetano se skalirati kako bi zadovoljio rastuće zahtjeve. Kontinuirano praćenje, profiliranje i napori za optimizaciju su neophodni za održavanje performansi i odziva sistema kako se radno opterećenje i složenost povećavaju tokom vremena.

Testiranje i validacija tokova rada

Testiranje i validacija su ključni aspekti razvoja i održavanja sistema za inteligentnu orkestraciju toka rada. S obzirom na složenu prirodu tokova rada zasnovanih na vještačkoj inteligenciji, neophodno je osigurati da svaka komponenta funkcioniše prema očekivanjima, da se cjelokupni tok rada ponaša ispravno i da su odluke vještačke inteligencije tačne i pouzdane. U ovom odjeljku ćemo istražiti različite tehnike i razmatranja za testiranje i validaciju inteligentnih tokova rada.

Jedinično testiranje komponenti toka rada

Jedinično testiranje uključuje testiranje pojedinačnih komponenti toka rada u izolaciji kako bi se provjerila njihova ispravnost i robusnost. Prilikom jediničnog testiranja komponenti zasnovanih na vještačkoj inteligenciji, razmotrite sljedeće:

1. **Validacija ulaza**: Testirajte sposobnost komponente da rukuje različitim tipovima ulaza, uključujući validne i nevalidne podatke. Provjerite da komponenta elegantno rukuje graničnim slučajevima i pruža odgovarajuće poruke o greškama ili izuzetke.

2. **Verifikacija izlaza**: Potvrdite da komponenta proizvodi očekivani izlaz za dati skup ulaza. Uporedite stvarni izlaz sa očekivanim rezultatima kako biste osigurali ispravnost.

3. **Rukovanje greškama**: Testirajte mehanizme rukovanja greškama komponente simulirajući različite scenarije grešaka, kao što su nevažeći ulaz, nedostupnost resursa ili neočekivani izuzeci. Provjerite da komponenta hvata i rukuje greškama na odgovarajući način.

4. **Granični uslovi**: Testirajte ponašanje komponente pod graničnim uslovima, kao što su prazan ulaz, maksimalna veličina ulaza ili ekstremne vrijednosti. Osigurajte da komponenta rukuje ovim uslovima elegantno bez rušenja ili proizvodnje netačnih rezultata.

Evo primjera jediničnog testa za komponentu toka rada u Ruby-ju 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, `OrderValidator` komponenta se testira koristeći dva test slučaja: jedan za validnu narudžbu i drugi za nevalidnu narudžbu. Test slučajevi verificiraju da `validate` metoda vraća očekivanu boolean vrijednost baziranu na validnosti narudžbe.

Integraciono Testiranje Interakcija u Toku Rada

Integraciono testiranje se fokusira na verifikaciju interakcija i toka podataka između različitih komponenti toka rada. Osigurava da komponente rade zajedno bez problema i proizvode očekivane rezultate. Kada se radi integraciono testiranje inteligentnih tokova rada, uzmite u obzir sljedeće:

1. **Interakcija Komponenti**: Testirajte komunikaciju i razmjenu podataka između komponenti toka rada. Verificirajte da se izlaz jedne komponente ispravno prenosi kao ulaz u sljedeću komponentu u toku rada.

2. **Konzistentnost Podataka**: Osigurajte da podaci ostaju konzistentni i tačni dok prolaze kroz tok rada. Verificirajte da se transformacije podataka, kalkulacije i agregacije

izvršavaju ispravno.

3. **Propagacija Izuzetaka**: Testirajte kako se izuzeci i greške propagiraju i obrađuju kroz komponente toka rada. Verificirajte da su izuzeci uhvaćeni, zabilježeni i pravilno obrađeni kako bi se spriječio prekid toka rada.

4. **Asinhrono Ponašanje**: Ako tok rada uključuje asinhrone komponente ili paralelno izvršavanje, testirajte mehanizme koordinacije i sinhronizacije. Osigurajte da se tok rada ponaša ispravno u konkurentnim i asinhronim scenarijima.

Evo primjera integracionog testa za tok rada u Ruby-ju koristeći RSpec framework za testiranje:

```
RSpec.describe OrderProcessingWorkflow do

  let(:order) { build(:order) }

  it 'processes the order successfully' do
    expect(OrderValidator).to receive(:validate).and_return(true)
    expect(InventoryManager).to receive(:check_availability).and_return(true)
    expect(PaymentProcessor).to receive(:process_payment).and_return(true)
    expect(ShippingService).to receive(:schedule_shipping).and_return(true)

    workflow = OrderProcessingWorkflow.new(order)
    result = workflow.process

    expect(result).to be true
    expect(order.status).to eq('processed')
  end

end
```

U ovom primjeru, `OrderProcessingWorkflow` se testira provjeravanjem interakcija između različitih komponenti toka rada. Test postavlja očekivanja za ponašanje svake komponente i osigurava da tok rada uspješno obrađuje narudžbu, ažurirajući status narudžbe u skladu s tim.

Testiranje AI tačaka odlučivanja

Testiranje AI tačaka odlučivanja je ključno za osiguravanje tačnosti i pouzdanosti tokova rada zasnovanih na vještačkoj inteligenciji. Prilikom testiranja AI tačaka odlučivanja, uzmite u obzir sljedeće:

1. **Tačnost odlučivanja**: Provjerite da AI komponenta donosi tačne odluke na osnovu ulaznih podataka i treniranog modela. Uporedite AI odluke s očekivanim ishodima ili referentnim podacima.

2. **Granični slučajevi**: Testirajte ponašanje AI komponente u graničnim slučajevima i neuobičajenim scenarijima. Provjerite da AI komponenta elegantno upravlja ovim slučajevima i donosi razumne odluke.

3. **Pristrasnost i pravednost**: Procijenite AI komponentu na potencijalne pristrasnosti i osigurajte da donosi pravedne i nepristrasne odluke. Testirajte komponentu s raznolikim ulaznim podacima i analizirajte ishode za bilo kakve diskriminatorne obrasce.

4. **Objašnjivost**: Ako AI komponenta pruža objašnjenja ili obrazloženja za svoje odluke, provjerite tačnost i jasnoću objašnjenja. Osigurajte da su objašnjenja usklađena s osnovnim procesom donošenja odluka.

Evo primjera testiranja AI tačke odlučivanja u Ruby-ju koristeći RSpec okvir za testiranje:

```ruby
RSpec.describe FraudDetector do
  describe '#detect_fraud' do
    context 'when transaction is fraudulent' do
      let(:tx) do
        build(:transaction, amount: 10_000, location: 'High-Risk Country')
      end

      it 'returns true' do
        expect(subject.detect_fraud(tx)).to be true
      end
    end

    context 'when transaction is legitimate' do
      let(:tx) do
        build(:transaction, amount: 100, location: 'Low-Risk Country')
      end

      it 'returns false' do
        expect(subject.detect_fraud(tx)).to be false
      end
    end
  end
end
```

U ovom primjeru, `FraudDetector` AI komponenta je testirana sa dva test slučaja: jedan za fraudulentnu transakciju i drugi za legitimnu transakciju. Test slučajevi verificiraju da metoda `detect_fraud` vraća očekivanu boolean vrijednost na osnovu karakteristika transakcije.

Testiranje od početka do kraja

Testiranje od početka do kraja uključuje testiranje cijelog toka rada od početka do završetka, simulirajući scenarije iz stvarnog svijeta i korisničke interakcije. Osigurava da se tok rada ponaša ispravno i proizvodi željene rezultate. Prilikom izvođenja testiranja od početka do kraja za inteligentne tokove rada, uzmite u obzir sljedeće:

1. **Korisnički scenariji**: Identificirajte uobičajene korisničke scenarije i testirajte

ponašanje toka rada u tim scenarijima. Verificirajte da tok rada ispravno obrađuje korisničke unose, donosi odgovarajuće odluke i proizvodi očekivane rezultate.

2. **Validacija podataka**: Osigurajte da tok rada validira i čisti korisničke unose kako bi se spriječile nedosljednosti u podacima ili sigurnosne ranjivosti. Testirajte tok rada s različitim vrstama ulaznih podataka, uključujući validne i nevalidne podatke.

3. **Oporavak od grešaka**: Testirajte sposobnost toka rada da se oporavi od grešaka i izuzetaka. Simulirajte scenarije grešaka i verificirajte da tok rada elegantno upravlja njima, bilježi greške i poduzima odgovarajuće akcije oporavka.

4. **Performanse i skalabilnost**: Procijenite performanse i skalabilnost toka rada pod različitim uslovima opterećenja. Testirajte tok rada s velikim brojem istovremenih zahtjeva i mjerite vrijeme odziva, korištenje resursa i ukupnu stabilnost sistema.

Evo primjera testa od početka do kraja za tok rada u Ruby-ju koristeći RSpec okvir za testiranje i Capybara biblioteku za simulaciju korisničkih interakcija:

```
RSpec.describe 'Order Processing Workflow' do
  scenario 'User places an order successfully' do
    visit '/orders/new'
    fill_in 'Product', with: 'Sample Product'
    fill_in 'Quantity', with: '2'
    fill_in 'Shipping Address', with: '123 Main St'
    click_button 'Place Order'

    expect(page).to have_content('Order Placed Successfully')
    expect(Order.count).to eq(1)
    expect(Order.last.status).to eq('processed')
  end
end
```

U ovom primjeru, testiranje s kraja na kraj simulira korisnika koji postavlja narudžbu putem web interfejsa. Popunjava potrebna polja obrasca, šalje narudžbu i provjerava da li je narudžba uspješno obrađena, prikazujući odgovarajuću poruku potvrde i ažurirajući status narudžbe u bazi podataka.

Kontinuirana integracija i isporuka

Da bi se osigurala pouzdanost i održivost inteligentnih tokova rada, preporučuje se integracija testiranja i validacije u pipeline kontinuirane integracije i isporuke (CI/CD). Ovo omogućava automatizirano testiranje i validaciju promjena u toku rada prije nego što se implementiraju u produkciju. Razmotrite sljedeće prakse:

1. **Automatsko izvršavanje testova**: Konfigurirajte CI/CD pipeline tako da automatski pokreće set testova kad god se naprave izmjene u kodu toka rada. Ovo osigurava da se bilo kakve regresije ili greške otkriju rano u procesu razvoja.

2. **Praćenje pokrivenosti testovima**: Mjerite i pratite pokrivenost testovima komponenti toka rada i AI tačaka odlučivanja. Težite visokoj pokrivenosti testovima kako biste osigurali da su kritične putanje i scenariji temeljito testirani.

3. **Kontinuirana povratna informacija**: Integrirajte rezultate testova i metrike kvaliteta koda u proces razvoja. Pružajte kontinuiranu povratnu informaciju programerima o statusu testova, kvalitetu koda i bilo kojim problemima otkrivenim tokom CI/CD procesa.

4. **Staging okruženja**: Implementirajte tok rada u staging okruženja koja vjerno odražavaju produkcijsko okruženje. Izvršite dodatno testiranje i validaciju u staging okruženju kako biste otkrili bilo kakve probleme vezane za infrastrukturu, konfiguraciju ili integraciju podataka.

5. **Mehanizmi za povrat**: Implementirajte mehanizme za povrat u slučaju neuspjeha implementacije ili kritičnih problema otkrivenih u produkciji. Osigurajte da se tok rada može brzo vratiti na prethodnu stabilnu verziju kako bi se minimiziralo vrijeme zastoja i utjecaj na korisnike.

Uključivanjem testiranja i validacije kroz cijeli životni ciklus razvoja inteligentnih tokova rada, organizacije mogu osigurati pouzdanost, tačnost i održivost svojih AI

sistema. Redovno testiranje i validacija pomažu u otkrivanju grešaka, sprječavanju regresija i izgradnji povjerenja u ponašanje i rezultate toka rada.

Dio 2: Obrasci

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Inženjerstvo promptova

Lanac razmišljanja

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Kako funkcioniše

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Primjeri

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Generiranje sadržaja

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Kreiranje Strukturiranih Entiteta

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Vođenje LLM Agenata

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Prednosti i razmatranja

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Prebacivanje režima

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Kako funkcioniše

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Kada ga koristiti

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Primjer

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Dodjeljivanje Uloge

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Kako Funkcioniše

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Kada ga Koristiti

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Primjeri

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Promptni Objekt

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Kako Funkcioniše

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Predložak Upita

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Kako Funkcionira

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Prednosti i Razmatranja

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Kada Ga Koristiti:

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Primjer

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Strukturirani Ulaz/Izlaz

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Kako Funkcioniše

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Skaliranje strukturiranog ulaza/izlaza

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Prednosti i razmatranja

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Ulančavanje upita

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Kako funkcionira

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Kada ga koristiti

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Primjer: Olympijin proces uvođenja

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Prepisivač Promptova

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Kako Funkcioniše

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Primjer

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Response Fencing

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Kako Funkcionira

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Prednosti i Razmatranja

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Rukovanje greškama

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Analizator upita

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Kako funkcioniše

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Implementacija

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Označavanje vrsta riječi (POS) i Prepoznavanje imenovanih entiteta (NER)

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Klasifikacija namjere

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Ekstrakcija ključnih riječi

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Prednosti

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Prepisivač upita

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Kako funkcioniše

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Primjer

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Prednosti

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Trbuhozborac

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Kako Funkcioniše

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Kada ga Koristiti

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Primjer

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Diskretne komponente

Predikat

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Kako funkcioniše

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Kada ga Koristiti

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Primjer

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

API Fasada

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Kako Funkcionira

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Ključne Prednosti

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Kada Je Koristiti

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Primjer

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Autentifikacija i Autorizacija

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Upravljanje Zahtjevima

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Formatiranje Odgovora

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Upravljanje Greškama i Granični Slučajevi

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Razmatranja o Skalabilnosti i Performansama

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Poređenje sa Drugim Dizajn Paternima

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Interpreter rezultata

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Kako funkcioniše

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Kada ga koristiti

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Primjer

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Virtuelna Mašina

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Kako Funkcioniše

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Kada ga Koristiti

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Primjer

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Iza Magije

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Specifikacija i Testiranje

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Specificiranje Ponašanja

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Pisanje Test Slučajeva

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Primjer: Testiranje Translator Komponente

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Ponavljanje HTTP Interakcija

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Čovjek u petlji (HITL)

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Obrasci visokog nivoa

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Hibridna inteligencija

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Adaptivni odgovor

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Izmjena uloga između ljudi i vještačke inteligencije

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Eskalacija

Kako funkcioniše

Ključne prednosti

Primjena u stvarnom svijetu: Zdravstvo

Povratna petlja

Kako to funkcioniše

Primjene i Primjeri

Napredne Tehnike u Integraciji Ljudskih Povratnih Informacija

Pasivno zračenje informacija

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Kako funkcioniše

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Kontekstualni prikaz informacija

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Proaktivne notifikacije

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Objašnjavajući uvidi

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Interaktivno istraživanje

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Ključne prednosti

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Primjene i primjeri

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Kolaborativno donošenje odluka (CDM)

Kako to funkcioniše

Primjer

Kontinuirano učenje

Kako funkcioniše

Primjene i primjeri

Primjer

Etička razmatranja

Uloga HITL-a u ublažavanju AI rizika

Tehnološki napredak i buduće perspektive

Izazovi i ograničenja HITL sistema

Inteligentno upravljanje greškama

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Tradicionalni pristupi upravljanju greškama

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Kontekstualna dijagnoza grešaka

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Kako funkcioniše

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Inženjerstvo promptova za kontekstualnu dijagnozu grešaka

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Generiranje potpomognuto pretraživanjem za kontekstualnu dijagnozu grešaka

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Inteligentno izvještavanje o greškama

Prediktivno sprječavanje grešaka

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Kako Funkcionira

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Pametni Oporavak od Grešaka

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Kako Funkcionira

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Personalizirana komunikacija o greškama

Kako funkcioniše

Adaptivni tok rada za upravljanje greškama

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Kako to funkcioniše

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Kontrola kvaliteta

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Eval

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Problem

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Rješenje

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Kako radi

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Primjer

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Razmatranja

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Razumijevanje zlatnih referenci

Kako rade evaluacije bez reference

Zaštitni mehanizam

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Problem

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Rješenje

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Kako to funkcioniše

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Primjer

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Razmatranja

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Zaštitne ograde i Evaluacije: Dvije strane istog novčića

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Međusobna zamjenjivost zaštitnih ograda i evaluacija bez reference

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Implementacija dvojnih zaštitnih ograda i evaluacija

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Pojmovnik

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Pojmovnik

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

A

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

B

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

C

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

D

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

E

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

F

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

G

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

H

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

I

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

J

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

K

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

L

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

M

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

N

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

O

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

P

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Q

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

R

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

S

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

T

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

U

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

V

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

W

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Z

Ovaj sadržaj nije dostupan u uzorku knjige. Knjiga se može kupiti na Leanpub-u na http://leanpub.com/patterns-of-application-development-using-ai-bs.

Index