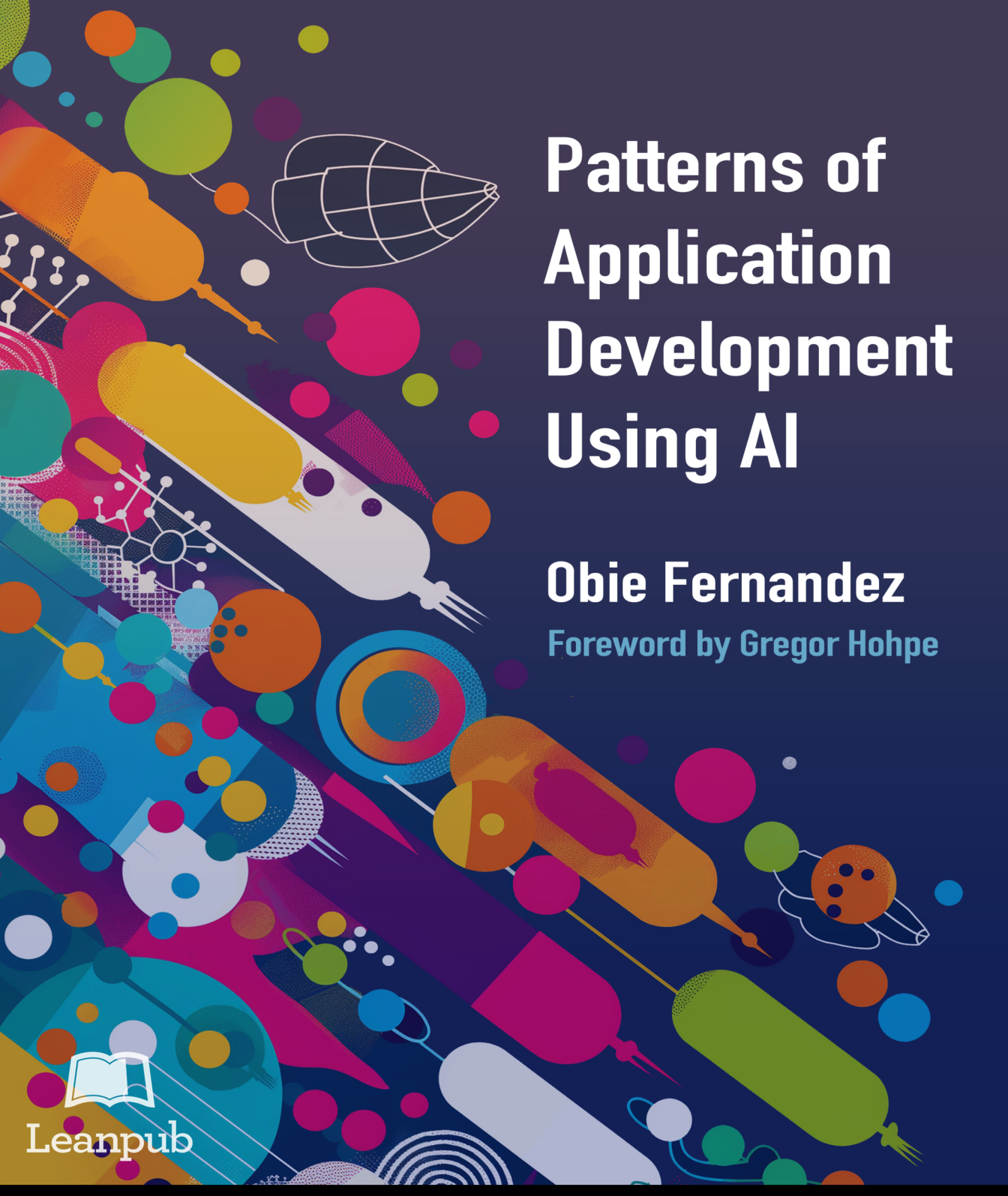


Ediție în Limba Română

Tipare în dezvoltarea aplicaţiilor folosind IA (Romanian Edition)

Obie Fernandez

Această carte se poate achiziționa de pe
http://leanpub.com/patterns-of-application-development-using-ai-ro

Această verisiune ese publicată pe 2025-01-23

Anunţă pe Tweeter despre carte!

Ajută autorul Obie Fernandez vorbind despre această carte pe Twitter!

Mesajul de Twitter sugerat pentru această carte este #poaduai.

Află ce spune lumea despre carte accesând următorul link de Twitter:

#poaduai

Pentru neînfricata mea regină, muza mea, lumina și iubirea mea, Victoria

Tot de 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

Cuprins

Partea 1: Abordări și Tehnici Fundamentale . 38

Cuvânt înainte de Gregor Hohpe

Prefaţă

Acest conţinut nu este disponibil în cartea de eşantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Despre Carte

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Despre Exemplele de Cod

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Ce Nu Acopăr

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Pentru Cine Este Această Carte

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Construirea unui Vocabular Comun

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Implicare

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Mulţumiri

Acest conţinut nu este disponibil în cartea de eşantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Ce e cu ilustraţiile?

Acest conţinut nu este disponibil în cartea de eşantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

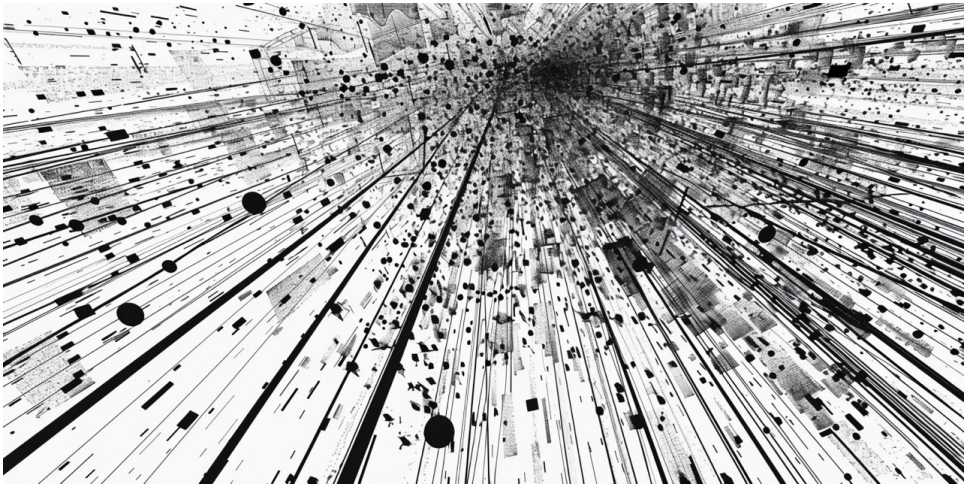
Despre Lean Publishing

Acest conţinut nu este disponibil în cartea de eşantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Despre Autor

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Introducere



Dacă sunteți nerăbdători să începeți să integrați Modele de Limbaj de Mari Dimensiuni (LLM) în proiectele voastre de programare, puteți trece direct la modelele și exemplele de cod prezentate în capitolele următoare. Cu toate acestea, pentru a aprecia pe deplin puterea și potențialul acestor modele, merită să acordăm un moment pentru a înțelege contextul mai larg și abordarea coerentă pe care o reprezintă.

Modelele nu sunt doar o colecție de tehnici izolate, ci mai degrabă un cadru unificat pentru integrarea AI în aplicațiile voastre. Eu folosesc Ruby on Rails, dar aceste modele ar trebui să funcționeze în practic orice alt mediu de programare. Ele abordează o gamă largă de aspecte, de la gestionarea datelor și optimizarea performanței până la experiența utilizatorului și securitate, oferind un set complet de instrumente pentru îmbunătățirea practicilor tradiționale de programare cu capacitățile AI.

Fiecare categorie de modele abordează o provocare sau o oportunitate specifică care apare atunci când încorporați componente AI în aplicația voastră. Înțelegând relațiile și

sinergiile dintre aceste modele, puteți lua decizii informate cu privire la unde și cum să aplicați AI în cel mai eficient mod.

Modelele nu sunt niciodată soluții prescriptive și nu ar trebui tratate ca atare. Ele sunt menite să fie blocuri de construcție adaptabile care ar trebui personalizate în funcție de cerințele și constrângerile unice ale propriei voastre aplicații. Aplicarea cu succes a acestor modele (ca oricare altele din domeniul software) se bazează pe o înțelegere profundă a domeniului problemei, a nevoilor utilizatorilor și a arhitecturii tehnice generale a proiectului vostru.

Gânduri despre Arhitectura Software

Am început să programez în anii 1980 și am fost implicat în scena hacker, și nu mi-am pierdut niciodată mentalitatea de hacker, chiar și după ce am devenit programator profesionist. De la început, am avut întotdeauna o sceptică sănătoasă cu privire la valoarea pe care arhitecții software din turnurile lor de fildeș o aduceau de fapt.

Unul dintre motivele pentru care sunt personal atât de entuziasmat de schimbările aduse de acest nou val puternic de tehnologie AI este impactul său asupra a ceea ce considerăm decizii de *arhitectură software*. Acesta pune sub semnul întrebării noțiunile tradiționale despre ce constituie modul "corect" de a proiecta și implementa proiectele noastre software. De asemenea, pune sub semnul întrebării dacă arhitectura mai poate fi considerată în primul rând ca *părțile unui sistem care sunt greu de schimbat*, având în vedere că îmbunătățirea AI face mai ușor ca niciodată să schimbi orice parte a proiectului tău, în orice moment.

Poate că intrăm în anii de vârf ai abordării "post-moderne" în ingineria software. În acest context, post-modern se referă la o schimbare fundamentală de la paradigmele tradiționale, unde dezvoltatorii erau responsabili pentru scrierea și întreținerea fiecărei linii de cod. În schimb, îmbrățișează ideea de a delega sarcini, cum ar fi manipularea datelor, algoritmii complecși și chiar părți întregi ale logicii aplicației, către biblioteci

terțe și API-uri externe. Această schimbare post-modernă reprezintă o îndepărtare semnificativă de la înțelepciunea convențională de a construi aplicații de la zero și provoacă dezvoltatorii să își regândească rolul în procesul de dezvoltare.

Am crezut întotdeauna că programatorii buni scriu doar codul care este absolut necesar să fie scris, bazându-mă pe învățăturile lui Larry Wall și ale altor luminari hacker ca el. Prin minimizarea cantității de cod scris, ne putem mișca mai repede, reduce suprafața pentru bug-uri, simplifica mentenanța și îmbunătăți fiabilitatea generală a aplicațiilor lor. Mai puțin cod ne permite să ne concentrăm pe logica de business centrală și experiența utilizatorului, în timp ce delegăm alte sarcini către alte servicii.

Acum că sistemele bazate pe AI pot gestiona sarcini care anterior erau domeniul exclusiv al codului scris de oameni, ar trebui să putem fi și mai productivi și agili, cu un focus mai mare ca niciodată pe crearea valorii de business și a experienței utilizatorului.

Bineînțeles că există compromisuri în delegarea unor părți mari ale proiectului vostru către sisteme AI, cum ar fi pierderea potențială a controlului și necesitatea unor mecanisme robuste de monitorizare și feedback. De aceea este nevoie de un nou set de abilități și cunoștințe, inclusiv cel puțin o înțelegere fundamentală a modului în care funcționează AI.

Ce este un Model de Limbaj de Mari Dimensiuni?

Modelele de Limbaj de Mari Dimensiuni (LLM) sunt un tip de model de inteligență artificială care au câștigat o atenție semnificativă în ultimii ani, de la lansarea GPT-3 de către OpenAI în 2020. LLM-urile sunt proiectate să proceseze, să înțeleagă și să genereze limbaj uman cu o acuratețe și fluență remarcabile. În această secțiune, vom arunca o privire scurtă asupra modului în care funcționează LLM-urile și de ce sunt potrivite pentru construirea componentelor de sistem inteligente.

La bază, LLM-urile se bazează pe algoritmi de învățare profundă, în special rețele neuronale. Aceste rețele sunt compuse din noduri interconectate, sau neuroni, care

procesează și transmit informații. Arhitectura preferată pentru LLM-uri este adesea modelul Transformer, care s-a dovedit a fi foarte eficient în gestionarea datelor secvențiale precum textul.

Modelele transformer se bazează pe mecanismul de atenție și sunt utilizate în principal pentru sarcini care implică date secvențiale, cum ar fi procesarea limbajului natural. Transformerii procesează datele de intrare simultan, nu secvențial, ceea ce le permite să capteze mai eficient dependențele pe distanțe lungi. Acestea au straturi de mecanisme de atenție care ajută modelul să se concentreze pe diferite părți ale datelor de intrare pentru a înțelege contextul și relațiile.

Procesul de antrenare pentru modelele lingvistice mari (LLM) implică expunerea modelului la cantități vaste de date textuale, cum ar fi cărți, articole, site-uri web și depozite de cod. În timpul antrenării, modelul învață să recunoască tipare, relații și structuri în text. Acesta captează proprietățile statistice ale limbajului, cum ar fi regulile gramaticale, asocierile dintre cuvinte și semnificațiile contextuale.

Una dintre tehnicile cheie utilizate în antrenarea LLM-urilor este învățarea nesupervizată. Acest lucru înseamnă că modelul învață din date fără etichetare sau îndrumare explicită. Descoperă singur tipare și reprezentări analizând co-apariția cuvintelor și frazelor în datele de antrenare. Acest lucru permite LLM-urilor să dezvolte o înțelegere profundă a limbajului și a subtilităților sale.

Un alt aspect important al LLM-urilor este capacitatea lor de a gestiona *contextul*. Când procesează un text, LLM-urile iau în considerare nu doar cuvintele individuale, ci și contextul înconjurător. Ele țin cont de cuvintele, propozițiile și chiar paragrafele anterioare pentru a înțelege sensul și intenția textului. Această înțelegere contextuală permite LLM-urilor să genereze răspunsuri coerente și relevante. Una dintre principalele modalități prin care evaluăm capacitățile unui model LLM dat este prin considerarea dimensiunii contextului pe care îl pot lua în considerare pentru a genera răspunsuri.

Odată antrenate, LLM-urile pot fi utilizate pentru o gamă largă de sarcini lingvistice. Acestea pot genera text asemănător celui uman, pot răspunde la întrebări, pot rezuma

documente, pot traduce limbi și pot chiar scrie cod. Versatilitatea LLM-urilor le face valoroase pentru construirea componentelor de sistem inteligent care pot interacționa cu utilizatorii, procesa și analiza date text și genera rezultate semnificative.

Prin încorporarea LLM-urilor în arhitectura aplicației, puteți crea componente AI care înțeleg și procesează input-ul utilizatorului, generează conținut dinamic și oferă recomandări sau acțiuni inteligente. Dar lucrul cu LLM-uri necesită o analiză atentă a cerințelor de resurse și a compromisurilor de performanță. LLM-urile necesită resurse computaționale intensive și pot avea nevoie de putere de procesare și memorie semnificative (cu alte cuvinte, bani) pentru a funcționa. Majoritatea dintre noi va trebui să evalueze implicațiile costurilor integrării LLM-urilor în aplicațiile noastre și să acționeze în consecință.

Înţelegerea Inferenţei

Inferența se referă la procesul prin care un model generează predicții sau rezultate bazate pe date noi, nevăzute anterior. Este faza în care modelul antrenat este utilizat pentru a lua decizii sau pentru a genera text, imagini sau alt conținut ca răspuns la inputurile utilizatorilor.

În timpul fazei de antrenare, un model AI învață dintr-un set mare de date ajustându-și parametrii pentru a minimiza eroarea în predicțiile sale. Odată antrenat, modelul poate aplica ce a învățat la date noi. Inferența este modul în care modelul folosește tiparele și cunoștințele învățate pentru a genera rezultate.

Pentru LLM-uri, inferența implică preluarea unui prompt sau text de intrare și producerea unui răspuns coerent și contextual relevant, sub forma unui flux de *tokeni* (despre care vom vorbi în curând). Aceasta ar putea însemna răspunsul la o întrebare, completarea unei propoziții, generarea unei poveşti sau traducerea unui text, printre multe alte sarcini.

Spre deosebire de modul în care gândim noi, "gândirea" unui model AI prin inferență se întâmplă într-o singură operație fără stare. Adică, gândirea sa este limitată la procesul său de generare. Literalmente trebuie să gândească cu voce tare, ca și cum ți-aș pune o întrebare și aș accepta un răspuns de la tine doar în stilul "flux al conștiinței".

Modelele Lingvistice Mari Vin în Multe Dimensiuni şi Varietăţi

În timp ce practic toate modelele lingvistice mari (LLM) populare se bazează pe aceeaşi arhitectură transformer de bază și sunt antrenate pe seturi uriaşe de date text, acestea vin într-o varietate de dimensiuni și sunt rafinate pentru diferite scopuri. Dimensiunea unui LLM, măsurată prin numărul de parametri din rețeaua sa neuronală, are un impact major asupra capacităților sale. Modelele mai mari cu mai mulți parametri, precum GPT-4, despre care se zvonește că ar avea 1-2 trilioane de parametri, sunt în general mai cunoscătoare și mai capabile decât modelele mai mici. Cu toate acestea, modelele mai mari necesită și mai multă putere de calcul pentru a rula, ceea ce se traduce prin costuri mai mari când le utilizați prin apeluri API.

Pentru a face LLM-urile mai practice și adaptate pentru cazuri de utilizare specifice, modelele de bază sunt adesea rafinate pe seturi de date mai țintite. De exemplu, un LLM poate fi antrenat pe un corpus mare de dialog pentru a-l specializa pentru AI conversațional. Altele sunt antrenate pe cod pentru a le îmbogăți cu cunoștințe de programare. Există chiar modele care sunt special antrenate pentru interacțiuni de tip joc de rol cu utilizatorii!

Modele bazate pe regăsire vs Modele generative

În lumea modelelor lingvistice mari (MLM), există două abordări principale pentru generarea răspunsurilor: modelele bazate pe regăsire și modelele generative. Fiecare

abordare are propriile puncte forte și limitări, iar înțelegerea diferențelor dintre ele te poate ajuta să alegi modelul potrivit pentru cazul tău specific de utilizare.

Modele bazate pe regăsire

Modelele bazate pe regăsire, cunoscute și sub numele de modele de regăsire a informațiilor, generează răspunsuri prin căutarea într-o bază de date mare de text preexistent și selectarea celor mai relevante pasaje bazate pe interogarea primită. Aceste modele nu generează text nou de la zero, ci mai degrabă îmbină fragmente din baza de date pentru a forma un răspuns coerent.

Unul dintre principalele avantaje ale modelelor bazate pe regăsire este capacitatea lor de a furniza informații precise și actualizate. Deoarece se bazează pe o bază de date de text curată, pot extrage informații relevante din surse de încredere și le pot prezenta utilizatorului. Acest lucru le face potrivite pentru aplicații care necesită răspunsuri precise și factuale, cum ar fi sistemele de întrebări și răspunsuri sau bazele de cunoștințe.

Cu toate acestea, modelele bazate pe regăsire au unele limitări. Ele sunt doar la fel de bune ca baza de date în care caută, astfel că calitatea și acoperirea bazei de date influențează direct performanța modelului. În plus, aceste modele pot avea dificultăți în generarea unor răspunsuri coerente și naturale, fiind limitate la textul disponibil în baza de date.

Nu acoperim utilizarea modelelor pure de regăsire în această carte.

Modele Generative

Modelele generative, pe de altă parte, creează text nou de la zero bazându-se pe tiparele și relațiile învățate în timpul antrenamentului. Aceste modele își folosesc înțelegerea limbajului pentru a genera răspunsuri originale care sunt adaptate la promptul de intrare.

Principalul punct forte al modelelor generative este capacitatea lor de a produce text creativ, coerent și relevant contextual. Ele pot participa la conversații deschise, pot

genera povești și pot chiar scrie cod. Acest lucru le face ideale pentru aplicații care necesită interacțiuni mai deschise și dinamice, cum ar fi chatboții, crearea de conținut și asistenții pentru scriere creativă.

Cu toate acestea, modelele generative pot produce uneori informații inconsistente sau factual incorecte, deoarece se bazează pe tiparele învățate în timpul antrenamentului și nu pe o bază de date curată de fapte. De asemenea, pot fi mai predispuse la prejudecăți și halucinații, generând text plauzibil dar nu neapărat adevărat.

Exemple de MLM generative includ seria GPT de la OpenAI (GPT-3, GPT-4) și Claude de la Anthropic.

Modele Hibride

Mai multe MLM-uri disponibile comercial combină atât abordările de regăsire, cât și cele generative într-un model hibrid. Aceste modele folosesc tehnici de regăsire pentru a găsi informații relevante dintr-o bază de date și apoi folosesc tehnici generative pentru a sintetiza acele informații într-un răspuns coerent.

Modelele hibride își propun să combine acuratețea factuală a modelelor bazate pe regăsire cu capacitățile de generare a limbajului natural ale modelelor generative. Ele pot furniza informații mai fiabile și actualizate, menținând în același timp capacitatea de a participa la conversații deschise.

Când alegi între modelele bazate pe regăsire și cele generative, ar trebui să iei în considerare cerințele specifice ale aplicației tale. Dacă obiectivul principal este să furnizezi informații precise și factuale, un model bazat pe regăsire poate fi cea mai bună alegere. Dacă aplicația necesită interacțiuni mai deschise și creative, un model generativ poate fi mai potrivit. Modelele hibride oferă un echilibru între cele două abordări și pot fi o alegere bună pentru aplicații care necesită atât acuratețe factuală, cât și generare de limbaj natural.

În cele din urmă, alegerea între modelele bazate pe regăsire și cele generative depinde de cazul specific de utilizare și de compromisurile între acuratețe, creativitate și

flexibilitate. Înţelegând punctele forte şi limitările fiecărei abordări, poţi lua decizii informate.

Ajustarea prin Instrucţiuni

Ajustarea prin instrucțiuni este un subset al antrenamentului conversațional care folosește prompturi și răspunsuri scrise de oameni pentru a face MLM-urile mai bune în înțelegerea și urmarea instrucțiunilor. Iată câteva exemple simple pentru a ilustra cum ajustarea prin instrucțiuni pe prompturi și răspunsuri scrise de oameni face MLM-urile mai bune în înțelegerea și urmarea instrucțiunilor:

Exemplul 1: Suport pentru Clienţi

Instrucțiune: "Oferă un răspuns unui client care cere o rambursare."

Prompt Scris de Om: "Aș dori să solicit o rambursare pentru achiziția mea recentă. Produsul nu a fost pe măsura așteptărilor mele."

Răspuns Scris de Om: "Îmi pare rău să aud că produsul nu a fost pe măsura așteptărilor tale. Pentru a procesa rambursarea, te rog să furnizezi numărul comenzii și motivul returnării. Vom analiza cererea ta și îți vom răspunde în 2-3 zile lucrătoare."

Exemplul 2: Asistenţă Educaţională

Instrucțiune: "Explică conceptul de fotosinteză unui copil de 10 ani."

Prompt scris de om: "Poți explica cum își fac plantele hrana?"

Răspuns scris de om: "Sigur! Plantele își fac hrana printr-un proces numit fotosinteză. Ele folosesc lumina soarelui, apa și dioxidul de carbon pentru a crea hrană și oxigen. Frunzele plantei au o parte specială numită clorofilă care le ajută să capteze lumina soarelui."

Exemplul 3: Asistent Personal

Instrucțiune: "Programează o întâlnire pentru marțea viitoare la ora 15:00."

> Prompt scris de om: “Poți programa o întâlnire pentru marți la ora 15:00?”
>
> Răspuns scris de om: “Sigur, am programat o întâlnire pentru marțea viitoare la ora 15:00. Mai ai nevoie de altceva?”

Rezultatul este un ecosistem divers de modele de limbaj de mari dimensiuni (LLM) cu diferite mărimi și specializări. Modelele mai mici, în intervalul de 1-7 miliarde de parametri, oferă capacități lingvistice generale bune, fiind în același timp mai eficiente în funcționare.

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

Modelele de dimensiune medie, cu aproximativ 30-70 miliarde de parametri, oferă capacități mai puternice de raționament și de urmărire a instrucțiunilor.

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

Când alegeți un LLM pentru a-l incorpora într-o aplicație, trebuie să echilibrați capacitățile modelului cu factori practici precum costul, latența, lungimea contextului și filtrarea conținutului. Modelele mai mici, antrenate pe instrucțiuni, sunt adesea cea mai bună alegere pentru sarcini lingvistice mai simple, în timp ce modelele mai mari pot fi necesare pentru raționament sau analiză complexă. Datele de antrenare ale modelului sunt, de asemenea, o considerație importantă, deoarece determină data limită a cunoștințelor modelului.

Anumite modele, cum ar fi unele de la Perplexity, sunt conectate la surse de informații în timp real, astfel încât nu au efectiv nicio dată limită. Când le puneți întrebări, acestea pot decide independent să facă căutări web și să preia pagini web arbitrare pentru a genera un răspuns.

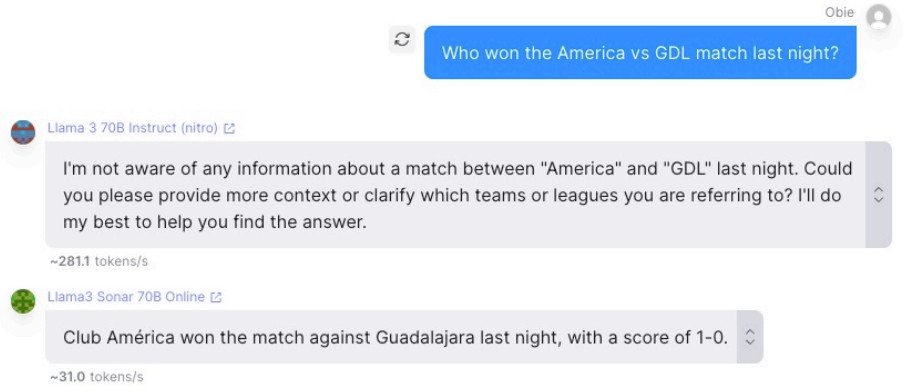


Figura 1. Llama3 cu și fără acces online

În cele din urmă, nu există un LLM universal. Înțelegerea variațiilor în dimensiunea modelului, arhitectură și antrenare este esențială pentru selectarea modelului potrivit pentru un caz de utilizare dat. Experimentarea cu diferite modele este singura modalitate practică de a descoperi care dintre ele oferă cea mai bună performanță pentru sarcina respectivă.

Tokenizarea: Împărţirea Textului în Bucăţi

Înainte ca un model de limbaj de mari dimensiuni să poată procesa text, acel text trebuie să fie împărțit în unități mai mici numite *tokeni*. Tokenii pot fi cuvinte individuale, părți ale cuvintelor sau chiar caractere singulare. Procesul de împărțire a textului în tokeni se numește tokenizare, și este un pas crucial în pregătirea datelor pentru un model de limbaj.

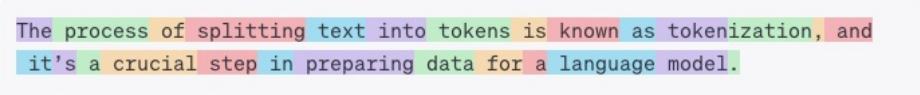


Figura 2. Această propoziție conține 27 de tokeni

Diferite LLM-uri folosesc strategii diferite de tokenizare, care pot avea un impact semnificativ asupra performanței și capacităților modelului. Unii tokenizatori comuni utilizați de LLM-uri includ:

- **GPT (Codificare prin Perechi de Biți)**: Tokenizatorii GPT folosesc o tehnică numită codificare prin perechi de biți (BPE) pentru a împărți textul în unități sub-cuvânt. BPE combină iterativ cele mai frecvente perechi de biți dintr-un corpus de text, formând un vocabular de tokeni sub-cuvânt. Acest lucru permite tokenizatorului să gestioneze cuvinte rare și noi prin împărțirea lor în bucăți sub-cuvânt mai comune. Tokenizatorii GPT sunt utilizați de modele precum GPT-3 și GPT-4.

- **Llama (SentencePiece)**: Tokenizatorii Llama utilizează biblioteca SentencePiece, care este un tokenizator și detokenizator de text nesupervizat. SentencePiece tratează textul de intrare ca o secvență de caractere Unicode și învață un vocabular de subsecvențe de cuvinte bazat pe un corpus de antrenament. Poate procesa orice limbă care poate fi codificată în Unicode, făcându-l potrivit pentru modele multilingve. Tokenizatorii Llama sunt utilizați de modele precum Llama și Alpaca de la Meta.

- **SentencePiece (Unigram)**: Tokenizatorii SentencePiece pot utiliza, de asemenea, un algoritm diferit numit Unigram, care se bazează pe o tehnică de regularizare a subsecvențelor de cuvinte. Tokenizarea Unigram determină vocabularul optimal de subsecvențe bazat pe un model de limbaj unigram, care atribuie probabilități unităților individuale de subsecvențe. Această abordare poate produce subsecvențe cu un sens semantic mai relevant în comparație cu BPE.

SentencePiece cu Unigram este utilizat de modele precum T5 și BERT de la Google.

- **Google Gemini (Tokenizare Multimodală)**: Google Gemini utilizează o schemă de tokenizare concepută pentru a gestiona diverse tipuri de date, inclusiv text, imagini, audio, videoclipuri și cod. Această capacitate multimodală permite lui Gemini să proceseze și să integreze diferite forme de informații. În mod special, Google Gemini 1.5 Pro are o fereastră de context care poate gestiona milioane de tokeni, mult mai mare decât modelele anterioare. Această fereastră de context extinsă permite modelului să proceseze un context mai mare, putând duce la răspunsuri mai precise. Cu toate acestea, este important de menționat că schema de tokenizare a Gemini este mult mai apropiată de un token per caracter decât alte modele. Acest lucru înseamnă că costul real al utilizării modelelor Gemini poate fi semnificativ mai mare decât te-ai aștepta dacă ești obișnuit să utilizezi modele precum GPT, deoarece prețurile Google se bazează pe caractere și nu pe tokeni.

Alegerea tokenizatorului afectează mai multe aspecte ale unui LLM, inclusiv:

- **Dimensiunea vocabularului**: Tokenizatorul determină dimensiunea vocabularului modelului, care este setul de tokeni unici pe care îi recunoaște. Un vocabular mai mare și mai detaliat poate ajuta modelul să gestioneze o gamă mai largă de cuvinte și fraze și chiar să devină multimodal (capabil să înțeleagă și să genereze mai mult decât doar text), dar crește și cerințele de memorie și complexitatea computațională a modelului.
- **Gestionarea cuvintelor rare și necunoscute**: Tokenizatorii care utilizează unități de subsecvențe, precum BPE și SentencePiece, pot descompune cuvintele rare și necunoscute în subsecvențe mai comune. Acest lucru permite modelului să facă presupuneri educate despre sensul cuvintelor pe care nu le-a văzut înainte, bazându-se pe subsecvențele pe care le conțin.

- **Suport multilingv**: Tokenizatorii precum SentencePiece, care pot gestiona orice limbă codificabilă în Unicode, sunt potriviți pentru modele multilingve care trebuie să proceseze text în mai multe limbi.

Când se alege un LLM pentru o anumită aplicație, este important să se ia în considerare tokenizatorul pe care îl utilizează și cât de bine se aliniază cu nevoile specifice de procesare a limbajului pentru sarcina respectivă. Tokenizatorul poate avea un impact semnificativ asupra capacității modelului de a gestiona terminologia specifică domeniului, cuvintele rare și textul multilingv.

Dimensiunea Contextului: Câtă Informaţie Poate Folosi un Model de Limbaj în Timpul Inferenţei?

Când discutăm despre modele de limbaj, dimensiunea contextului se referă la cantitatea de text pe care un model o poate lua în considerare când procesează sau generează răspunsurile sale. Este, în esență, o măsură a cantității de informații pe care modelul o poate "ține minte" și utiliza pentru a-și informa ieșirile (exprimată în tokeni). Dimensiunea contextului unui model de limbaj poate avea un impact semnificativ asupra capacităților sale și asupra tipurilor de sarcini pe care le poate efectua eficient.

Ce Este Dimensiunea Contextului?

În termeni tehnici, dimensiunea contextului este determinată de numărul de tokeni (cuvinte sau părți de cuvinte) pe care un model de limbaj îi poate procesa într-o singură secvență de intrare. Acest lucru este adesea denumit "capacitatea de atenție" sau "fereastra de context" a modelului. Cu cât dimensiunea contextului este mai mare, cu atât mai mult text poate lua în considerare modelul simultan când generează un răspuns sau efectuează o sarcină.

Diferite modele de limbaj au dimensiuni de context variate, de la câteva sute de tokeni până la milioane de tokeni. Ca referință, un paragraf tipic de text poate conține

aproximativ 100-150 de tokeni, în timp ce o carte întreagă ar putea conține zeci sau sute de mii de tokeni.

> Există chiar cercetări privind metode eficiente de scalare a Modelelor de Limbaj de Mari Dimensiuni (LLM) bazate pe Transformers pentru intrări infinit de lungi cu memorie și calcul limitat.

De ce este importantă dimensiunea contextului?

Dimensiunea contextului unui model lingvistic are un impact semnificativ asupra capacității sale de a înțelege și genera text coerent și relevant contextual. Iată câteva motive cheie pentru care dimensiunea contextului contează:

1. **Înțelegerea conținutului de lungă durată**: Modelele cu dimensiuni mai mari ale contextului pot înțelege și analiza mai bine texte mai lungi, cum ar fi articole, rapoarte sau chiar cărți întregi. Acest lucru este crucial pentru sarcini precum sumarizarea documentelor, răspunsul la întrebări și analiza conținutului.

2. **Menținerea coerenței**: O fereastră de context mai mare permite modelului să mențină coerența și consistența pe porțiuni mai lungi de text generat. Acest lucru este important pentru sarcini precum generarea de povești, sisteme de dialog și crearea de conținut, unde menținerea unei narațiuni sau a unui subiect consistent este esențială. Este, de asemenea, absolut crucial când se utilizează LLM-uri pentru generarea sau transformarea datelor structurate.
3. **Captarea dependențelor pe distanță lungă**: Unele sarcini lingvistice necesită înțelegerea relațiilor dintre cuvinte sau fraze care sunt departe unele de altele în text. Modelele cu dimensiuni mai mari ale contextului sunt mai bine echipate pentru a capta aceste dependențe pe distanță lungă, care pot fi importante pentru sarcini precum analiza sentimentelor, traducerea și înțelegerea limbajului.

4. **Gestionarea instrucțiunilor complexe**: În aplicațiile unde modelele lingvistice sunt utilizate pentru a urma instrucțiuni complexe, cu mai mulți pași, o dimensiune mai mare a contextului permite modelului să ia în considerare întregul set de instrucțiuni când generează un răspuns, în loc să se bazeze doar pe ultimele câteva cuvinte.

Exemple de modele lingvistice cu diferite dimensiuni ale contextului

Iată câteva exemple de modele lingvistice cu diferite dimensiuni ale contextului:

- OpenAI GPT-3.5 Turbo: 4.095 tokens
- Mistral 7B Instruct: 32.768 tokens
- Anthropic Claude v1: 100.000 tokens
- OpenAI GPT-4 Turbo: 128.000 tokens
- Anthropic Claude v2: 200.000 tokens
- Google Gemini Pro 1.5: 2,8M tokens

După cum puteți vedea, există o gamă largă de dimensiuni ale contextului între aceste modele, de la aproximativ 4.000 de tokens pentru modelul OpenAI GPT-3.5 Turbo până la 200.000 de tokens pentru modelul Anthropic Claude v2. Unele modele, precum Google PaLM 2 și OpenAI GPT-4, oferă diferite variante cu dimensiuni mai mari ale contextului (de exemplu, versiuni "32k"), care pot gestiona secvențe de input și mai lungi. Iar în prezent (aprilie 2024), Google Gemini Pro se laudă cu aproape 3 milioane de tokens!

Merită menționat că dimensiunea contextului poate varia în funcție de implementarea specifică și versiunea unui anumit model. De exemplu, modelul original OpenAI GPT-4 are o dimensiune a contextului de 8.191 tokens, în timp ce variantele ulterioare ale GPT-4, precum Turbo și 4o, au o dimensiune mult mai mare a contextului de 128.000 tokens.

> Sam Altman a comparat limitările actuale ale contextului cu kilobiții de memorie de lucru cu care programatorii de calculatoare personale trebuiau să se descurce în anii '80 și a spus că în viitorul apropiat vom putea încadra "toate datele personale" în contextul unui model lingvistic de mari dimensiuni.

Alegerea dimensiunii potrivite a contextului

Când selectați un model lingvistic pentru o anumită aplicație, este important să luați în considerare cerințele privind dimensiunea contextului pentru sarcina respectivă. Pentru sarcinile care implică texte scurte, izolate, precum analiza sentimentelor sau răspunsul la întrebări simple, o dimensiune mai mică a contextului poate fi suficientă. Cu toate acestea, pentru sarcinile care necesită înțelegerea și generarea unor texte mai lungi și mai complexe, va fi probabil necesară o dimensiune mai mare a contextului.

Merită menționat că dimensiunile mai mari ale contextului vin adesea cu costuri computaționale crescute și timpi de procesare mai lenți, deoarece modelul trebuie să ia în considerare mai multe informații când generează un răspuns. Ca atare, trebuie să găsiți un echilibru între dimensiunea contextului și performanță când alegeți un model lingvistic pentru aplicația dumneavoastră.

> De ce să nu alegem pur și simplu modelul cu cea mai mare dimensiune a contextului și să-l încărcăm cu cât mai multe informații posibile? Ei bine, pe lângă factorii de performanță, celălalt considerent principal este costul. În martie 2024, un *singur* ciclu prompt-răspuns folosind Google Gemini Pro 1.5 cu un context complet vă va costa aproape 8 dolari (USD). Dacă aveți un caz de utilizare care justifică această cheltuială,

> mai multă putere vouă! Dar pentru majoritatea aplicațiilor, este pur și simplu prea scump cu ordine de mărime.

Găsirea Acelor în Căpiţe de Fân

Conceptul de a găsi un ac în carul cu fân a fost mult timp o metaforă pentru provocările recuperării în seturi mari de date. În domeniul modelelor mari de limbaj, adaptăm puțin această analogie. Imaginați-vă că nu căutăm doar un singur fapt îngropat într-un text vast (precum o antologie completă de eseuri ale lui Paul Graham), ci multiple fapte împrăștiate peste tot. Acest scenariu seamănă mai mult cu găsirea mai multor ace într-un câmp întins, nu doar într-o singură căpiță de fân. Iată surpriza: nu doar că trebuie să localizăm aceste ace, dar trebuie să le și împletim într-un fir coerent.

Când sunt puse în fața sarcinii de a recupera și raționa despre multiple fapte încorporate în contexte lungi, modelele mari de limbaj se confruntă cu o dublă provocare. În primul rând, există problema directă a acurateței recuperării—aceasta scade în mod natural pe măsură ce numărul de fapte crește. Este de așteptat; la urma urmei, urmărirea mai multor detalii într-un text vast pune la încercare chiar şi cele mai sofisticate modele.

În al doilea rând, şi poate mai critic, este provocarea raționamentului cu aceste fapte. Este una să extragi fapte; este cu totul altceva să le sintetizezi într-o narațiune sau un răspuns coerent. Aici intervine adevăratul test. Performanţa modelelor mari de limbaj în sarcinile de raționament tinde să se degradeze mai mult decât în sarcinile simple de recuperare. Această degradare nu ține doar de volum; este vorba despre dansul intricate între context, relevanță și inferență.

De ce se întâmplă acest lucru? Ei bine, să luăm în considerare dinamica memoriei şi atenției în cogniția umană, care sunt oglindite într-o oarecare măsură în modelele mari de limbaj. Când procesează cantități mari de informații, modelele mari de limbaj, ca și oamenii, pot pierde urma detaliilor anterioare pe măsură ce absorb altele noi. Acest lucru

este valabil mai ales în cazul modelelor care nu sunt proiectate explicit să prioritizeze sau să reviziteze automat segmentele anterioare de text.

Mai mult, capacitatea unui model mare de limbaj de a împleti aceste fapte recuperate într-un răspuns coerent este similară cu construirea narativă. Acest lucru necesită nu doar o recuperare a informației, ci o înțelegere profundă și o plasare contextuală, care rămâne o provocare dificilă pentru AI-ul actual.

Așadar, ce înseamnă asta pentru noi ca dezvoltatori și integratori ai acestor tehnologii? Trebuie să fim profund conștienți de aceste limitări când proiectăm sisteme care se bazează pe modele mari de limbaj pentru a gestiona sarcini complexe, cu text lung. Înțelegerea faptului că performanța ar putea să se degradeze în anumite condiții ne ajută să stabilim așteptări realiste și să construim mecanisme mai bune de rezervă sau strategii suplimentare.

Modalităţi: Dincolo de Text

În timp ce majoritatea modelelor de limbaj de astăzi sunt concentrate pe procesarea și generarea de text, există o tendință crescândă către modele multimodale care pot în mod nativ să primească și să producă multiple tipuri de date, precum imagini, audio și video. Aceste modele multimodale deschid noi posibilități pentru aplicații bazate pe AI care pot înțelege și genera conținut în diferite modalități.

Ce sunt Modalităţile?

În contextul modelelor de limbaj, modalitățile se referă la diferitele tipuri de date pe care un model le poate procesa și genera. Cea mai comună modalitate este textul, care include limbaj scris în diverse forme precum cărți, articole, site-uri web și postări pe rețelele sociale. Cu toate acestea, există mai multe alte modalități care sunt din ce în ce mai mult încorporate în modelele de limbaj:

- **Imagini**: Date vizuale precum fotografii, ilustrații și diagrame.

- **Audio**: Date sonore precum vorbire, muzică și sunete ambientale.
- **Video**: Date vizuale în mișcare, adesea însoțite de audio, precum clipuri video și filme.

Fiecare modalitate prezintă provocări și oportunități unice pentru modelele de limbaj. De exemplu, imaginile necesită ca modelul să înțeleagă concepte și relații vizuale, în timp ce audio necesită ca modelul să proceseze și să genereze vorbire și alte sunete.

Modele de Limbaj Multimodale

Modelele de limbaj multimodale sunt proiectate să gestioneze multiple modalități într-un singur model. Aceste modele au de obicei componente sau straturi specializate care pot atât să înțeleagă intrările, cât și să genereze date de ieșire în diferite modalități. Câteva exemple notabile de modele de limbaj multimodale includ:

- **OpenAI's GPT-4o**: GPT-4o este un model mare de limbaj care înțelege și procesează în mod nativ audio vocal pe lângă text. Această capacitate permite GPT-4o să efectueze sarcini precum transcrierea limbajului vorbit, generarea de text din intrări audio și furnizarea de răspunsuri bazate pe întrebări vocale.
- **OpenAI's GPT-4 cu intrare vizuală**: GPT-4 este un model mare de limbaj care poate procesa atât text, cât și imagini. Când i se oferă o imagine ca intrare, GPT-4 poate analiza conținutul imaginii și poate genera text care descrie sau răspunde la informația vizuală.
- **Google's Gemini**: Gemini este un model multimodal care poate gestiona text, imagini și video. Folosește o arhitectură unificată care permite înțelegerea și generarea între modalități, făcând posibile sarcini precum descrierea imaginilor, sumarizarea videoclipurilor și răspunsul la întrebări vizuale.

- **DALL-E și Stable Diffusion**: Deși nu sunt modele lingvistice în sensul tradițional, aceste modele demonstrează puterea IA multimodale prin generarea de imagini din descrieri textuale. Ele evidențiază potențialul modelelor care pot traduce între diferite modalități.

Beneficii şi Aplicaţii ale Modelelor Multimodale

Modelele lingvistice multimodale oferă mai multe beneficii şi permit o gamă largă de aplicaţii, inclusiv:

- **Înțelegere îmbunătățită**: Prin procesarea informațiilor din multiple modalități, aceste modele pot obține o înțelegere mai cuprinzătoare a lumii, similar cu modul în care oamenii învață din diverse intrări senzoriale.
- **Generare inter-modală**: Modelele multimodale pot genera conţinut într-o modalitate bazată pe input dintr-o altă modalitate, cum ar fi crearea unei imagini dintr-o descriere text sau generarea unui rezumat video dintr-un articol scris.
- **Accesibilitate**: Modelele multimodale pot face informația mai accesibilă prin traducerea între modalități, cum ar fi generarea descrierilor text ale imaginilor pentru utilizatorii cu deficiențe de vedere sau crearea versiunilor audio ale conţinutului scris.
- **Aplicații creative**: Modelele multimodale pot fi utilizate pentru sarcini creative precum generarea de artă, muzică sau videoclipuri bazate pe comenzi textuale, deschizând noi posibilități pentru artişti şi creatori de conţinut.

Pe măsură ce modelele lingvistice multimodale continuă să avanseze, acestea vor juca probabil un rol din ce în ce mai important în dezvoltarea aplicațiilor bazate pe IA care pot înțelege şi genera conţinut în multiple modalități. Acest lucru va permite interacțiuni mai naturale şi intuitive între oameni şi sistemele de IA, precum şi va debloca noi posibilități pentru exprimarea creativă şi diseminarea cunoştinţelor.

Ecosisteme ale Furnizorilor

Când vine vorba de încorporarea modelelor lingvistice mari (MLM) în aplicații, aveți la dispoziție o gamă tot mai mare de opțiuni din care să alegeți. Fiecare furnizor major de MLM, precum OpenAI, Anthropic, Google și Cohere, oferă propriul său ecosistem

de modele, API-uri și instrumente. Alegerea furnizorului potrivit implică luarea în considerare a diferiților factori, inclusiv prețuri, performanță, filtrarea conținutului, confidențialitatea datelor și opțiunile de personalizare.

OpenAI

OpenAI este unul dintre cei mai cunoscuți furnizori de MLM, seria sa GPT (GPT-3, GPT-4) fiind utilizată pe scară largă în diverse aplicații. OpenAI oferă un API ușor de utilizat care vă permite să integrați cu ușurință modelele lor în aplicații. Ei oferă o gamă de modele cu capacități și prețuri diferite, de la modelul de bază Ada până la puternicul model Davinci.

Ecosistemul OpenAI include și instrumente precum OpenAI Playground, care vă permite să experimentați cu prompturi și să ajustați fin modelele pentru cazuri specifice de utilizare. Ei oferă opțiuni de filtrare a conținutului pentru a ajuta la prevenirea generării de conținut inadecvat sau dăunător.

Când folosesc modelele OpenAI direct, mă bazez pe biblioteca ruby-openai a lui Alex Rudall.

Anthropic

Anthropic este un alt jucător major în domeniul MLM, modelele lor Claude câștigând popularitate pentru performanța puternică și considerentele etice. Anthropic se concentrează pe dezvoltarea sistemelor de IA sigure și responsabile, cu un accent puternic pe filtrarea conținutului și evitarea rezultatelor dăunătoare.

Ecosistemul Anthropic include API-ul Claude, care vă permite să integrați modelul în aplicațiile lor, precum și instrumente pentru ingineria prompturilor și ajustare fină. Ei oferă și modelul Claude Instant, care încorporează capacități de căutare web pentru răspunsuri mai actualizate și factuale.

Când folosesc modelele Anthropic direct, mă bazez pe biblioteca anthrophic a lui Alex Rudall.

Google

Google a dezvoltat mai multe MLM-uri puternice, inclusiv Gemini, BERT, T5 și PaLM. Aceste modele sunt cunoscute pentru performanța lor puternică într-o gamă largă de sarcini de procesare a limbajului natural. Ecosistemul Google include bibliotecile TensorFlow și Keras, care oferă instrumente și cadre pentru construirea și antrenarea modelelor de învățare automată.

Google oferă, de asemenea, o platformă Cloud AI, care vă permite să implementați și să scalați cu ușurință modelele lor în cloud. Ei oferă o gamă de modele pre-antrenate și API-uri pentru sarcini precum analiza sentimentelor, recunoașterea entităților și traducere.

Meta

Meta, cunoscută anterior ca Facebook, este profund implicată în dezvoltarea modelelor lingvistice mari, evidențiată prin lansarea modelelor precum LLaMA și OPT. Aceste modele se remarcă prin performanța lor puternică în diverse sarcini lingvistice și sunt disponibile în mare parte prin canale open-source, susținând angajamentul Meta față de cercetare și colaborare comunitară.

Ecosistemul Meta este construit în principal în jurul PyTorch, o bibliotecă de învățare automată open-source preferată pentru capacitățile sale computaționale dinamice și flexibilitate, facilitând cercetarea și dezvoltarea inovatoare în IA.

Pe lângă ofertele lor tehnice, Meta pune un accent puternic pe dezvoltarea etică a inteligenței artificiale. Ei implementează o filtrare robustă a conținutului și se concentrează pe reducerea prejudecăților, aliniate cu obiectivele lor mai largi de siguranță și responsabilitate în aplicațiile AI.

Cohere

Cohere este un nou venit în domeniul LLM, concentrându-se pe a face LLM-urile mai accesibile și mai ușor de utilizat decât competitorii. Ecosistemul lor include API-ul

Cohere, care oferă acces la o gamă de modele pre-antrenate pentru sarcini precum generarea de text, clasificare și sumarizare.

Cohere oferă, de asemenea, instrumente pentru ingineria prompturilor, ajustare fină și filtrarea conținutului. Ei pun accent pe confidențialitatea și securitatea datelor, cu funcționalități precum stocarea criptată a datelor și controlul accesului.

Ollama

Ollama este o platformă auto-găzduită care permite utilizatorilor să gestioneze și să implementeze diverse modele de limbaj de mari dimensiuni (LLM) local pe propriile mașini, oferindu-le control complet asupra modelelor lor de AI fără a depinde de servicii cloud externe. Această configurație este ideală pentru cei care prioritizează confidențialitatea datelor și doresc să-și gestioneze operațiunile AI intern.

Platforma suportă o gamă de modele, inclusiv versiuni de Llama, Phi, Gemma și Mistral, care variază în dimensiune și cerințe computaționale. Ollama face ușoară descărcarea și rularea acestor modele direct din linia de comandă folosind comenzi simple precum `ollama run <model_name>`, și este proiectată să funcționeze pe diferite sisteme de operare, inclusiv macOS, Linux și Windows.

Pentru dezvoltatorii care doresc să integreze modele open-source în aplicațiile lor fără a utiliza un API la distanță, Ollama oferă un CLI pentru gestionarea ciclurilor de viață ale modelelor similar cu instrumentele de gestionare a containerelor. De asemenea, suportă configurații și prompturi personalizate, permițând un grad ridicat de personalizare pentru a adapta modelele la nevoi sau cazuri de utilizare specifice.

Ollama este în mod special potrivită pentru utilizatorii cu cunoștințe tehnice și dezvoltatori datorită interfeței sale în linia de comandă și flexibilității pe care o oferă în gestionarea și implementarea modelelor AI. Acest lucru o face un instrument puternic pentru companii și indivizi care necesită capabilități AI robuste fără a compromite securitatea și controlul.

Platforme Multi-Model

În plus, există furnizori care găzduiesc o varietate largă de modele open-source, precum Together.ai şi Groq.. Aceste platforme oferă flexibilitate şi personalizare, permiţându-vă să rulaţi şi, în unele cazuri, chiar să ajustaţi fin modelele open-source conform nevoilor specifice. De exemplu, Together.ai oferă acces la o gamă de LLM-uri open-source, permiţând utilizatorilor să experimenteze cu diferite modele şi configuraţii. Groq se concentrează pe furnizarea unei completări de înaltă performanţă care, la momentul scrierii acestei cărţi, pare aproape magică

Alegerea unui Furnizor LLM

Când alegeţi un furnizor LLM, ar trebui să luaţi în considerare factori precum:

- **Preţuri**: Diferiţi furnizori oferă modele de preţuri diferite, de la plată per utilizare până la planuri bazate pe abonament. Este important să luaţi în considerare utilizarea preconizată şi bugetul când selectaţi un furnizor.
- **Performanţă**: Performanţa LLM-urilor poate varia semnificativ între furnizori, aşa că este important să testaţi şi să evaluaţi modelele pe cazuri de utilizare specifice înainte de a lua o decizie.
- **Filtrarea Conţinutului**: În funcţie de aplicaţie, filtrarea conţinutului poate fi o consideraţie critică. Unii furnizori oferă opţiuni mai robuste de filtrare a conţinutului decât alţii.
- **Confidenţialitatea Datelor**: Dacă aplicaţia manipulează date sensibile ale utilizatorilor, este important să alegeţi un furnizor cu practici puternice de confidenţialitate şi securitate a datelor.
- **Personalizare**: Unii furnizori oferă mai multă flexibilitate în ceea ce priveşte ajustarea fină şi personalizarea modelelor pentru cazuri specifice de utilizare.

În cele din urmă, alegerea furnizorului LLM depinde de cerinţele şi constrângerile specifice ale aplicaţiei. Prin evaluarea atentă a opţiunilor şi luarea în considerare

a factorilor precum prețurile, performanța și confidențialitatea datelor, puteți selecta furnizorul care se potrivește cel mai bine nevoilor dumneavoastră.

Este, de asemenea, important de menționat că peisajul LLM este în constantă evoluție, cu noi furnizori și modele apărând în mod regulat. Ar trebui să vă mențineți la curent cu cele mai recente dezvoltări și să fiți deschiși la explorarea noilor opțiuni pe măsură ce devin disponibile.

OpenRouter

Pe parcursul acestei cărți mă voi baza exclusiv pe OpenRouter ca furnizorul meu preferat de API. Motivul este simplu: este un loc unic pentru toate cele mai populare modele comerciale și open-source. Dacă sunteți nerăbdători să începeți să programați cu AI, unul dintre cele mai bune locuri pentru a începe este cu propria mea Bibliotecă Ruby OpenRouter.

Gânduri despre Performanță

Când incorporăm modele de limbaj în aplicații, performanța este o considerație critică. Performanța unui model de limbaj poate fi măsurată în termeni de *latență* (timpul necesar pentru a genera un răspuns) și *capacitate de procesare* (numărul de cereri pe care le poate gestiona pe unitate de timp).

Timpul până la Primul Token (TTFT) este o altă metrică esențială de performanță, deosebit de relevantă pentru chatboți și aplicații care necesită răspunsuri interactive, în timp real. TTFT măsoară latența din momentul în care cererea utilizatorului este primită până în momentul în care primul cuvânt (sau token) al răspunsului este generat. Această metrică este crucială pentru menținerea unei experiențe de utilizare fluide și angajante, deoarece răspunsurile întârziate pot duce la frustrarea și dezangajarea utilizatorului.

Aceste metrici de performanță pot avea un impact semnificativ asupra experienței utilizatorului și a scalabilității aplicației.

Mai mulți factori pot influența performanța unui model de limbaj, inclusiv:

Numărul de Parametri: Modelele mai mari cu mai mulți parametri necesită în general mai multe resurse computaționale și pot avea o latență mai mare și o capacitate de procesare mai mică în comparație cu modelele mai mici.

Hardware: Performanța unui model de limbaj poate varia semnificativ în funcție de hardware-ul pe care rulează. Furnizorii de cloud oferă instanțe GPU și TPU optimizate pentru sarcini de învățare automată, care pot accelera semnificativ inferența modelului.

Unul dintre lucrurile frumoase despre OpenRouter este că pentru multe dintre modelele pe care le oferă, ai posibilitatea de a alege între furnizori de cloud cu diferite profiluri de performanță și costuri.

Cuantizare: Tehnicile de cuantizare pot fi utilizate pentru a reduce amprenta de memorie și cerințele computaționale ale unui model prin reprezentarea ponderilor și activărilor cu tipuri de date de precizie mai mică. Acest lucru poate îmbunătăți performanța fără a sacrifica semnificativ calitatea. Ca dezvoltator de aplicații, probabil nu te vei implica în antrenarea propriilor modele la diferite niveluri de cuantizare, dar este bine să fii cel puțin familiarizat cu terminologia.

Procesare în loturi: Procesarea mai multor cereri simultan în loturi poate îmbunătăți capacitatea de procesare prin amortizarea costurilor de încărcare a modelului și transfer de date.

Memorare în cache: Memorarea în cache a rezultatelor prompturilor sau secvențelor de intrare frecvent utilizate poate reduce numărul de cereri de inferență și îmbunătăți performanța generală.

Când selectăm un model de limbaj pentru o aplicație în producție, este important să măsurăm performanța acestuia pe sarcini și configurații hardware reprezentative. Acest lucru poate ajuta la identificarea potențialelor blocaje și asigurarea că modelul poate îndeplini obiectivele de performanță necesare.

De asemenea, merită să luăm în considerare compromisurile între performanța modelului și alți factori precum costul, flexibilitatea și ușurința integrării. De exemplu, utilizarea unui model mai mic, mai puțin costisitor, cu latență mai mică poate fi preferabilă pentru aplicații care necesită răspunsuri în timp real, în timp ce un model mai mare și mai puternic poate fi mai potrivit pentru procesarea în loturi sau sarcini de raționament complex.

Experimentarea cu Diferite Modele LLM

Alegerea unui LLM este rareori o decizie permanentă. Având în vedere că noi modele îmbunătățite sunt lansate în mod regulat, este bine să construim aplicații într-un mod modular care permite înlocuirea diferitelor modele de limbaj în timp. Prompturile și seturile de date pot fi adesea reutilizate între modele cu modificări minime. Acest lucru vă permite să profitați de cele mai recente progrese în modelarea limbajului fără a trebui să reproiectați complet aplicațiile.

Capacitatea de a comuta ușor între o gamă largă de modele este încă un motiv pentru care îmi place foarte mult OpenRouter.

Când facem upgrade la un nou model de limbaj, este important să testăm și să validăm temeinic performanța și calitatea rezultatelor pentru a ne asigura că îndeplinește cerințele aplicației. Acest lucru poate implica reantrenarea sau ajustarea fină a modelului pe date specifice domeniului, precum și actualizarea oricăror componente dependente de ieșirile modelului.

Prin proiectarea aplicațiilor cu performanța și modularitatea în minte, puteți crea sisteme scalabile, eficiente și pregătite pentru viitor care se pot adapta la peisajul în rapidă evoluție al tehnologiei de modelare a limbajului.

Sisteme AI Compuse

Înainte de a încheia introducerea noastră, merită menționat că înainte de 2023 și explozia interesului pentru AI generativ declanșată de ChatGPT, abordările tradiționale AI se bazau de obicei pe integrarea unor modele unice, închise. În contrast, *Sistemele AI Compuse* folosesc pipeline-uri complexe de componente interconectate care lucrează împreună pentru a obține un comportament inteligent.

În esență, sistemele AI compuse constau din multiple module, fiecare proiectat pentru a efectua sarcini sau funcții specifice. Aceste module pot include generatoare, sisteme de recuperare, sisteme de clasificare, clasificatoare și diverse alte componente specializate. Prin descompunerea sistemului general în unități mai mici și focalizate, dezvoltatorii pot crea arhitecturi AI mai flexibile, scalabile și mai ușor de întreținut.

Unul dintre avantajele cheie ale sistemelor AI compuse este capacitatea lor de a combina punctele forte ale diferitelor tehnici și modele AI. De exemplu, un sistem poate utiliza un model lingvistic de mari dimensiuni (LLM) pentru înțelegerea și generarea limbajului natural, în timp ce folosește un model separat pentru recuperarea informațiilor sau luarea deciziilor bazată pe reguli. Această abordare modulară vă permite să selectați cele mai bune instrumente și tehnici pentru fiecare sarcină specifică, în loc să vă bazați pe o soluție universală.

Cu toate acestea, construirea sistemelor AI compuse prezintă și provocări unice. În special, asigurarea coerenței și consecvenței generale a comportamentului sistemului necesită mecanisme robuste de testare, monitorizare și guvernanță.

Apariția LLM-urilor puternice precum GPT-4 ne permite să experimentăm cu sisteme AI compuse mai ușor ca niciodată, deoarece aceste modele avansate sunt capabile să gestioneze multiple roluri într-un sistem compus, cum ar fi clasificarea, ierarhizarea și generarea, pe lângă capacitățile lor de înțelegere a limbajului natural. Această versatilitate permite dezvoltatorilor să prototipeze și să itereze rapid arhitecturi AI compuse, deschizând noi posibilități pentru dezvoltarea aplicațiilor inteligente.

Modele de Implementare pentru Sisteme AI Compuse

Sistemele AI compuse pot fi implementate folosind diverse modele, fiecare conceput pentru a răspunde unor cerințe și cazuri de utilizare specifice. Să explorăm patru modele comune de implementare: Întrebare și Răspuns, Sisteme Multi-Agent/Agentice de Rezolvare a Problemelor, AI Conversațional și CoPiloți.

Întrebare şi Răspuns

Sistemele de Întrebare și Răspuns (Q&A) se concentrează pe furnizarea unei recuperări de informații îmbunătățită cu capacitățile de înțelegere ale modelelor AI pentru a funcționa ca mai mult decât un simplu motor de căutare. Prin combinarea modelelor lingvistice puternice cu surse externe de cunoştinţe folosind Generarea Augmentată prin Recuperare (RAG), sistemele de Întrebare și Răspuns evită halucinațiile și oferă răspunsuri precise și relevante contextual la întrebările utilizatorilor.

Componentele cheie ale unui sistem Q&A bazat pe LLM includ:

- **Înțelegerea și reformularea întrebărilor**: Analizarea întrebărilor utilizatorilor și reformularea acestora pentru a se potrivi mai bine cu sursele de cunoștințe subiacente.
- **Recuperarea cunoștințelor**: Recuperarea informațiilor relevante din surse de date structurate sau nestructurate pe baza întrebării reformulate.

- **Generarea răspunsurilor**: Generarea unor răspunsuri coerente și informative prin integrarea cunoștințelor recuperate cu capacitățile generative ale modelului lingvistic.

Subsistemele RAG sunt deosebit de importante în domeniile Q&A unde furnizarea de informații precise și actualizate este crucială, cum ar fi asistența clienți, managementul cunoștințelor sau aplicațiile educaționale

Sisteme Multi-Agent/Agentice de Rezolvare a Problemelor

Sistemele multi-agent, cunoscute și ca sisteme *Agentice*, constau în mai mulți agenți autonomi care lucrează împreună pentru a rezolva probleme complexe. Fiecare agent are un rol specific, un set de abilități și acces la instrumente sau surse de informații relevante. Prin colaborare și schimb de informații, acești agenți pot aborda sarcini care ar fi dificile sau imposibile pentru un singur agent să le gestioneze singur.

Principiile cheie ale sistemelor multi-agent de rezolvare a problemelor includ:

- **Specializare**: Fiecare agent se concentrează pe un aspect specific al problemei, folosindu-și capacitățile și cunoștințele unice.
- **Colaborare**: Agenții comunică și își coordonează acțiunile pentru a atinge un obiectiv comun, adesea prin transmiterea de mesaje sau memorie partajată.
- **Adaptabilitate**: Sistemul se poate adapta la condiții sau cerințe în schimbare prin ajustarea rolurilor și comportamentelor agenților individuali.

Sistemele multi-agent sunt potrivite pentru aplicații care necesită rezolvarea distribuită a problemelor, cum ar fi optimizarea lanțului de aprovizionare, managementul traficului sau planificarea răspunsului în situații de urgență

AI Conversaţional

Sistemele AI conversaționale permit interacțiuni în limbaj natural între utilizatori și agenți inteligenți. Aceste sisteme combină capacități de înțelegere a limbajului natural, gestionare a dialogului și generare a limbajului pentru a oferi experiențe conversaționale angajante și personalizate.

Componentele principale ale unui sistem AI conversațional includ:

- **Recunoașterea intenției**: Identificarea intenției utilizatorului pe baza input-ului său, cum ar fi adresarea unei întrebări, formularea unei cereri sau exprimarea unui sentiment.
- **Extragerea entităților**: Extragerea entităților sau parametrilor relevanți din input-ul utilizatorului, cum ar fi date, locații sau nume de produse.
- **Gestionarea dialogului**: Menținerea stării conversației, determinarea răspunsului adecvat bazat pe intenția și contextul utilizatorului și gestionarea interacțiunilor cu mai multe schimburi.
- **Generarea răspunsurilor**: Generarea unor răspunsuri asemănătoare celor umane folosind modele lingvistice, șabloane sau metode bazate pe recuperare.

Sistemele AI conversaționale sunt utilizate frecvent în roboți de conversație pentru serviciul clienți, asistenți virtuali și interfețe controlate vocal. După cum am menționat anterior, majoritatea abordărilor, modelelor și exemplelor de cod din această carte sunt extrase direct din munca mea la un sistem AI conversațional de mari dimensiuni numit Olympia

CoPiloţi

CoPiloții sunt asistenți bazați pe AI care lucrează alături de utilizatori umani pentru a le îmbunătăți productivitatea și capacitatea de luare a deciziilor. Aceste sisteme folosesc o combinație de procesare a limbajului natural, învățare automată și cunoștințe specifice

domeniului pentru a oferi recomandări inteligente, a automatiza sarcini și a oferi suport contextual.

Caracteristicile cheie ale CoPiloților includ:

- **Personalizare**: Adaptarea la preferințele individuale ale utilizatorilor, fluxurile de lucru și stilurile de comunicare.
- **Asistență proactivă**: Anticiparea nevoilor utilizatorilor și oferirea de sugestii sau acțiuni relevante fără solicitări explicite.
- **Învățare continuă**: Îmbunătățirea performanței în timp prin învățarea din feedback-ul utilizatorilor, interacțiuni și date.

CoPiloții sunt tot mai mult utilizați în diverse domenii, cum ar fi dezvoltarea software (de exemplu, completarea codului și detectarea erorilor), scrierea creativă (de exemplu, sugestii de conținut și editare), și analiza datelor (de exemplu, perspective și recomandări de vizualizare)

Aceste modele de implementare evidențiază versatilitatea și potențialul sistemelor AI compuse. Prin înțelegerea caracteristicilor și cazurilor de utilizare ale fiecărui model, puteți lua decizii informate atunci când proiectați și implementați aplicații inteligente. Deși această carte nu este în mod specific despre implementarea sistemelor AI compuse, multe, dacă nu toate abordările și modelele similare se aplică integrării componentelor AI discrete în dezvoltarea tradițională a aplicațiilor.

Roluri în Sistemele AI Compuse

Sistemele AI compuse sunt construite pe o fundație de module interconectate, fiecare conceput pentru a îndeplini un rol specific. Aceste module lucrează împreună pentru a crea comportamente inteligente și a rezolva probleme complexe. Este util să fiți familiarizați cu aceste roluri atunci când vă gândiți unde ați putea implementa sau înlocui părți ale aplicației dvs. cu componente AI discrete.

Generator

Generatoarele sunt responsabile pentru producerea de date sau conținut nou bazat pe modele învățate sau prompt-uri de intrare. Lumea AI are multe tipuri diferite de generatoare, dar în contextul modelelor de limbaj prezentate în această carte, generatoarele pot crea text asemănător celui uman, pot completa propoziții parțiale sau pot genera răspunsuri la întrebările utilizatorilor. Ele joacă un rol crucial în sarcini precum crearea de conținut, generarea de dialog și augmentarea datelor.

Sistem de Recuperare

Sistemele de recuperare sunt utilizate pentru a căuta și extrage informații relevante din seturi mari de date sau baze de cunoștințe. Acestea folosesc tehnici precum căutarea semantică, potrivirea cuvintelor cheie sau similaritatea vectorială pentru a găsi cele mai pertinente puncte de date bazate pe o interogare sau un context dat. Sistemele de recuperare sunt esențiale pentru sarcini care necesită acces rapid la informații specifice, cum ar fi răspunsurile la întrebări, verificarea faptelor sau recomandarea de conținut.

Sistem de Ordonare

Sistemele de ordonare sunt responsabile pentru ordonarea sau prioritizarea unui set de elemente bazate pe anumite criterii sau scoruri de relevanță. Acestea atribuie ponderi sau scoruri fiecărui element și apoi le sortează în consecință. Sistemele de ordonare sunt folosite frecvent în motoarele de căutare, sistemele de recomandare sau orice aplicație unde prezentarea celor mai relevante rezultate utilizatorilor este crucială.

Clasificator

Clasificatorii sunt utilizați pentru a categorisi sau eticheta puncte de date bazate pe clase sau categorii predefinite. Aceștia învață din date de antrenament etichetate și apoi prezic clasa unor noi instanțe nevăzute. Clasificatorii sunt fundamentali pentru sarcini precum

analiza sentimentelor, detectarea spam-ului sau recunoașterea imaginilor, unde scopul este de a atribui o categorie specifică fiecărei intrări.

Unelte şi Agenţi

Pe lângă aceste roluri fundamentale, sistemele AI compuse încorporează adesea unelte şi agenţi pentru a-şi îmbunătăți funcționalitatea şi adaptabilitatea:

- **Unelte**: Uneltele sunt componente software discrete sau API-uri care efectuează acțiuni sau calcule specifice. Acestea pot fi invocate de alte module, cum ar fi generatoarele sau sistemele de recuperare, pentru a îndeplini sub-sarcini sau a colecta informații suplimentare. Exemple de unelte includ motoare de căutare web, calculatoare sau biblioteci de vizualizare a datelor.
- **Agenți**: Agenții sunt entități autonome care pot percepe mediul lor, lua decizii şi întreprinde acțiuni pentru a atinge obiective specifice. Aceştia se bazează adesea pe o combinație de diferite tehnici AI, cum ar fi planificarea, raționamentul şi învățarea, pentru a opera eficient în condiții dinamice sau incerte. Agenții pot fi utilizați pentru a modela comportamente complexe sau pentru a coordona acțiunile mai multor module într-un sistem AI compus.

Într-un sistem AI compus pur, interacțiunea dintre aceste componente este orchestrată prin interfețe şi protocoale de comunicare bine definite. Datele curg între module, cu ieşirea unei componente servind ca intrare pentru alta. Această arhitectură modulară permite flexibilitate, scalabilitate şi mentenabilitate, deoarece componentele individuale pot fi actualizate, înlocuite sau extinse fără a afecta întregul sistem.

Prin valorificarea puterii acestor componente şi a interacțiunilor lor, sistemele AI compuse pot aborda probleme complexe din lumea reală care necesită o combinație de diferite capabilități AI. Pe măsură ce explorăm abordările şi modelele pentru integrarea AI în dezvoltarea aplicațiilor, țineți cont că aceleaşi principii şi tehnici utilizate în

sistemele AI compuse pot fi aplicate pentru a crea aplicații inteligente, adaptive și centrate pe utilizator.

În următoarele capitole din Partea 1, vom aprofunda abordările și tehnicile fundamentale pentru integrarea componentelor AI în procesul dvs. de dezvoltare a aplicațiilor. De la ingineria prompt-urilor și generarea augmentată cu recuperare până la date cu auto-vindecare și orchestrarea fluxurilor de lucru inteligente, vom acoperi o gamă largă de modele și bune practici pentru a vă ajuta să construiți aplicații de ultimă generație bazate pe AI.

Partea 1: Abordări şi Tehnici Fundamentale

Această parte a cărţii prezintă diferite modalităţi de integrare a utilizării AI în aplicaţiile tale. Capitolele acoperă o serie de abordări şi tehnici conexe, de la concepte de nivel mai înalt precum Restrângerea Căii şi Retrieval Augmented Generation, până la idei pentru programarea propriului strat de abstractizare peste API-urile de completare chat LLM.

Scopul acestei părţi a cărţii este să te ajute să înţelegi tipurile de comportament pe care le poţi implementa cu AI, înainte de a intra prea adânc în modelele specifice de implementare care reprezintă focusul Părţii 2.

Abordările din Partea 1 se bazează pe idei pe care le-am folosit în codul meu, modele clasice de arhitectură şi integrare a aplicaţiilor enterprise, plus metafore pe care le-am folosit când am explicat capacităţile AI-ului altor persoane, inclusiv părţilor interesate din business care nu au pregătire tehnică.

Îngustează Calea

“Îngustează calea” se referă la focalizarea AI-ului asupra sarcinii respective. O folosesc ca mantra ori de câte ori mă frustrez că AI-ul acționează “prostește” sau în moduri neașteptate. Mantra îmi amintește că eșecul este probabil vina mea și că ar trebui probabil să îngustez calea și mai mult.

Necesitatea îngustării căii provine din vastele cantități de cunoștințe conținute în modelele lingvistice mari, în special modelele de clasă mondială precum cele de la OpenAI și Anthropic care au literalmente trilioane de parametri.

Accesul la o asemenea gamă largă de cunoștințe este fără îndoială puternic și produce comportamente emergente precum teoria minții și abilitatea de a raționa în moduri asemănătoare oamenilor. Totuși, acest volum copleșitor de informații prezintă și provocări când vine vorba de generarea unor răspunsuri precise și exacte la prompturi specifice, mai ales dacă aceste prompturi sunt menite să exhibe un comportament deterministic care poate fi integrat cu dezvoltarea de software și algoritmii "normali".

Mai mulți factori duc la aceste provocări.

Supraîncărcarea cu Informații: Modelele lingvistice mari sunt antrenate pe cantități masive de date care acoperă diverse domenii, surse și perioade de timp. Această cunoaștere extensivă le permite să se angajeze în diverse subiecte și să genereze răspunsuri bazate pe o înțelegere largă a lumii. Totuși, când se confruntă cu un prompt specific, modelul ar putea să se chinuie să filtreze informațiile irelevante, contradictorii sau învechite/depășite, ducând la răspunsuri care duc lipsă de focalizare sau acuratețe. În funcție de ce încerci să faci, simpla cantitate de informații *contradictorii* disponibile modelului poate ușor să copleșească abilitatea sa de a furniza răspunsul sau comportamentul pe care îl cauți.

Ambiguitate Contextuală: Dat fiind vastul *spațiu latent* de cunoștințe, modelele lingvistice mari ar putea întâmpina ambiguitate când încearcă să înțeleagă *contextul* promptului tău. Fără o îngustare sau ghidare adecvată, modelul poate genera răspunsuri care sunt tangențial relaționate dar nu direct relevante pentru intențiile tale. Acest tip de eșec duce la răspunsuri care sunt în afara subiectului, inconsistente sau care nu adresează nevoile tale declarate. În acest caz, îngustarea căii se referă la *dezambiguizarea* contextului, asigurând că contextul pe care îl oferi face modelul să se concentreze doar pe informațiile cele mai relevante din cunoștințele sale de bază.

Notă: Când începi cu "ingineria prompturilor" este mult mai probabil să ceri modelului să facă lucruri fără să explici corespunzător rezultatul dorit; este nevoie de practică pentru a nu fi ambiguu!

Inconsistențe Temporale: Întrucât modelele lingvistice sunt antrenate pe date care au fost create în perioade diferite, ele pot poseda cunoștințe care sunt învechite, depășite sau nu mai sunt exacte. De exemplu, informațiile despre evenimente curente, descoperiri științifice sau avansuri tehnologice s-ar putea să fi evoluat de când datele de antrenare ale modelului au fost colectate. Fără îngustarea căii pentru a prioritiza surse mai recente și mai de încredere, modelul ar putea genera răspunsuri bazate pe informații învechite sau incorecte, ducând la inexactități și inconsistențe în rezultatele sale.

Nuanțe Specifice Domeniului: Diferite domenii și câmpuri au propriile terminologii specifice, convenții și baze de cunoștințe. Gândește-te la practic orice TLA (Acronim din Trei Litere) și vei realiza că majoritatea au mai mult de un înțeles. De exemplu, MSK poate face referire la Amazon's Managed Streaming for Apache Kafka, Memorial Sloan Kettering Cancer Center, sau sistemul MusculoSKeletal uman.

Când un prompt necesită expertiză într-un domeniu particular, cunoștințele generice ale unui model lingvistic mare ar putea să nu fie suficiente pentru a furniza răspunsuri exacte și nuanțate. Îngustarea căii prin focalizarea pe informații specifice domeniului, fie prin ingineria prompturilor sau generare augmentată prin recuperare, permite modelului să genereze răspunsuri care sunt mai aliniate cu cerințele și așteptările specifice domeniului tău.

Spațiul Latent: Incomprehensibil de Vast

Când menționez "spațiul latent" al unui model lingvistic, mă refer la vastul peisaj multidimensional de cunoștințe și informații pe care modelul l-a învățat în timpul procesului său de antrenare. Este ca un tărâm ascuns în interiorul rețelelor neuronale ale modelului, unde sunt stocate toate tiparele, asocierile și reprezentările limbajului.

Imaginează-ți că explorezi un teritoriu vast, necartografiat, plin de noduri interconectate nenumărate. Fiecare nod reprezintă o bucată de informație, un concept sau o relație pe care modelul a învățat-o. Pe măsură ce navighezi prin acest spațiu, vei descoperi că unele

noduri sunt mai apropiate între ele, indicând o conexiune puternică sau similaritate, în timp ce altele sunt mai depărtate, sugerând o relație mai slabă sau mai distantă.

Provocarea cu spațiul latent este că acesta este incredibil de complex și multidimensional. Gândiți-vă la el ca fiind la fel de imens precum universul nostru fizic, cu clustere de galaxii și distanțe vaste, de neimaginat, de spațiu gol între ele.

Pentru că conține mii de dimensiuni, spațiul latent nu poate fi observat sau interpretat direct de către oameni. Este o reprezentare abstractă pe care modelul o folosește intern pentru a procesa și genera limbaj. Când furnizați un prompt de intrare modelului, acesta mapează efectiv acel prompt într-o locație specifică în spațiul latent. Modelul folosește apoi informațiile și conexiunile înconjurătoare din acel spațiu pentru a genera un răspuns.

Problema este că modelul a învățat o cantitate enormă de informații din datele sale de antrenament, și nu toate sunt relevante sau precise pentru o sarcină dată. De aceea îngustarea căii devine atât de importantă. Prin furnizarea de instrucțiuni clare, exemple și context în prompturile dumneavoastră, practic ghidați modelul să se concentreze pe regiuni specifice din spațiul latent care sunt cele mai relevante pentru rezultatul dorit.

Un alt mod de a privi acest lucru este ca și cum ai folosi un reflector într-un muzeu complet întunecat. Dacă ați vizitat vreodată Luvrul sau Metropolitan Museum of Art, atunci acesta este tipul de scară despre care vorbesc. Spațiul latent este muzeul, plin de nenumărate obiecte și detalii. Promptul dumneavoastră este reflectorul, iluminând zone specifice și direcționând atenția modelului către cele mai importante informații. Fără această ghidare, modelul poate rătăci fără țintă prin spațiul latent, culegând informații irelevante sau contradictorii pe parcurs.

În timp ce lucrați cu modele lingvistice și vă creați prompturile, țineți cont de conceptul de spațiu latent. Obiectivul dumneavoastră este să navigați eficient prin acest vast peisaj al cunoașterii, ghidând modelul către informațiile cele mai relevante și precise pentru sarcina dumneavoastră. Prin îngustarea căii și oferirea unei ghidări clare, puteți debloca

întregul potențial al spațiului latent al modelului și genera răspunsuri de înaltă calitate și coerente.

În timp ce descrierile anterioare ale modelelor lingvistice și ale spațiului latent pe care îl navighează pot părea puțin magice sau abstracte, este important să înțelegem că prompturile nu sunt vrăji sau incantații. Modul în care funcționează modelele lingvistice se bazează pe principiile algebrei liniare și teoriei probabilităților.

În esență, modelele lingvistice sunt modele probabilistice ale textului, similar cu modul în care o curbă normală este un model statistic al datelor. Acestea sunt antrenate printr-un proces numit modelare auto-regresivă, unde modelul învață să prezică probabilitatea următorului cuvânt într-o secvență bazată pe cuvintele care îl preced. În timpul antrenamentului, modelul începe cu ponderi aleatorii și le ajustează treptat pentru a atribui probabilități mai mari textului care seamănă cu eșantioanele din lumea reală pe care a fost antrenat.

Cu toate acestea, gândirea modelelor lingvistice ca simple modele statistice, precum regresia liniară, nu oferă cea mai bună intuiție pentru înțelegerea comportamentului lor. O analogie mai potrivită este să ne gândim la ele ca la programe probabilistice, care sunt modele ce permit manipularea variabilelor aleatorii și pot reprezenta relații statistice complexe.

Programele probabilistice pot fi reprezentate prin modele grafice, care oferă o modalitate vizuală de a înțelege dependențele și relațiile dintre variabilele din model. Această perspectivă poate oferi informații valoroase despre funcționarea modelelor complexe de generare a textului precum GPT-4 și Claude.

În lucrarea "Language Model Cascades" de Dohan et al., autorii aprofundează detaliile despre cum programele probabilistice pot fi aplicate modelelor lingvistice. Ei arată cum acest cadru poate fi folosit pentru a înțelege comportamentul acestor modele și pentru a ghida dezvoltarea unor strategii de promptare mai eficiente.

O perspectivă cheie din această abordare probabilistică este că modelul lingvistic creează efectiv un portal către un univers alternativ unde documentele dorite există. Modelul

atribuie ponderi tuturor documentelor posibile bazate pe probabilitatea lor, îngustând efectiv spațiul posibilităților pentru a se concentra pe cele mai relevante.

Acest lucru ne aduce înapoi la tema centrală a "îngustării căii". Obiectivul principal al promptării este de a condiționa modelul probabilistic într-un mod care concentrează masa predicțiilor sale, focalizând pe informațiile sau comportamentul specific pe care dorim să-l obținem. Prin furnizarea de prompturi atent elaborate, putem ghida modelul să navigheze spațiul latent mai eficient și să genereze rezultate mai relevante și coerente.

Cu toate acestea, este important să ținem cont că modelul lingvistic este în cele din urmă limitat de informațiile cu care a fost antrenat. În timp ce poate genera text similar cu documente existente sau poate combina idei în moduri noi, nu poate inventa informații complet noi din nimic. De exemplu, nu putem aștepta ca modelul să ofere un leac pentru cancer dacă un astfel de leac nu a fost descoperit și documentat în datele sale de antrenament.

În schimb, puterea modelului constă în abilitatea sa de a găsi și sintetiza informații similare cu cele pe care le includem în prompt. Înțelegând natura probabilistică a acestor modele și modul în care prompturile pot fi folosite pentru a le condiționa rezultatele, putem valorifica mai eficient capacitățile lor pentru a genera perspective și conținut valoros.

Să analizăm prompturile de mai jos. În primul, "Mercury" folosit singur ar putea face referire la planetă, la elementul chimic, sau la zeul roman, dar cea mai probabilă variantă este planeta. Într-adevăr, GPT-4 oferă un răspuns lung care începe cu *Mercur este cea mai mică planetă și cea mai apropiată de Soare din Sistemul Solar....* Al doilea prompt face referire specific la elementul chimic. Al treilea se referă la figura mitologică romană, cunoscută pentru viteza sa și rolul său de mesager divin.

```
# Prompt 1
Tell me about: Mercury

# Prompt 2
Tell me about: Mercury element

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

Prin adăugarea doar a câtorva cuvinte suplimentare, am schimbat complet modul în care reacționează AI-ul. După cum veți învăța mai târziu în carte, trucurile sofisticate de inginerie a prompturilor, cum ar fi promptarea n-shot, intrarea/ieșirea structurată și Lanțul de Gândire sunt doar modalități inteligente de a condiționa ieșirea modelului.

Așadar, în cele din urmă, arta ingineriei prompturilor constă în înțelegerea modului de navigare prin vastul peisaj probabilistic al cunoștințelor modelului lingvistic pentru a restrânge calea către informația sau comportamentul specific pe care îl căutăm.

Pentru cititorii cu o înțelegere solidă a matematicii avansate, fundamentarea înțelegerii acestor modele pe principiile teoriei probabilităților și algebrei liniare vă poate ajuta cu siguranță! Pentru restul dintre voi care doriți să dezvoltați strategii eficiente pentru obținerea rezultatelor dorite, să ne menținem la abordări mai intuitive.

Cum Se "Îngustează" Calea

Pentru a aborda aceste provocări ale cunoștințelor prea vaste, folosim tehnici care ajută la ghidarea procesului de generare al modelului lingvistic și concentrează atenția acestuia asupra informațiilor cele mai relevante și precise.

Iată cele mai importante tehnici, în ordinea recomandată, adică ar trebui să încercați mai întâi Ingineria Prompturilor, apoi RAG, și în final, dacă este necesar, ajustarea fină.

Ingineria Prompturilor Abordarea fundamentală constă în crearea de prompturi care includ instrucțiuni specifice, constrângeri sau exemple pentru a ghida generarea

răspunsurilor modelului. Acest capitol acoperă fundamentele Ingineriei Prompturilor în secțiunea următoare, și acoperim multe modele specifice de inginerie a prompturilor în Partea 2 a cărții. Aceste modele includ Distilarea Prompturilor, o tehnică care se concentrează pe rafinarea și optimizarea prompturilor pentru a extrage ceea ce AI-ul consideră a fi informația cea mai relevantă și concisă.

Augmentarea Contextului. Recuperarea dinamică a informațiilor relevante din baze de cunoștințe externe sau documente pentru a furniza modelului un context focalizat în momentul promptării. Tehnicile populare de augmentare a contextului includ Generarea Augmentată prin Recuperare (RAG) Așa-numitele "modele online" precum cele furnizate de Perplexity pot să-și augmenteze contextul cu rezultate în timp real ale căutărilor pe internet.

În ciuda puterii lor, LLM-urile nu sunt antrenate pe seturile tale unice de date, care pot fi private sau specifice problemei pe care încerci să o rezolvi. Tehnicile de Augmentare a Contextului îți permit să oferi LLM-urilor acces la date din spatele API-urilor, în baze de date SQL sau blocate în PDF-uri și prezentări.

Ajustarea Fină sau Adaptarea pe Domeniu Antrenarea modelului pe seturi de date specifice domeniului pentru a-i specializa cunoștințele și capacitățile de generare pentru o anumită sarcină sau domeniu.

Reducerea Temperaturii

Temperatura este un *hiperparametru* utilizat în modelele lingvistice bazate pe transformeri care controlează aleatoriul și creativitatea textului generat. Este o valoare între 0 și 1, unde valorile mai mici fac ieșirea mai focalizată și deterministă, în timp ce valorile mai mari o fac mai diversă și impredictibilă.

Când temperatura este setată la 1, modelul lingvistic generează text bazat pe distribuția completă de probabilitate a următorului token, permițând răspunsuri mai creative și

variate. Totuși, acest lucru poate duce și la generarea de către model a unui text mai puțin relevant sau coerent.

Pe de altă parte, când temperatura este setată la 0, modelul lingvistic selectează întotdeauna tokenul cu cea mai mare probabilitate, "îngustându-și" efectiv calea. Aproape toate componentele mele AI folosesc o temperatură setată la sau aproape de 0, deoarece rezultă în răspunsuri mai focalizate și predictibile. Este absolut util când vrei ca modelul să *urmeze instrucțiuni*, să fie atent la funcțiile care i-au fost furnizate sau pur și simplu ai nevoie de răspunsuri mai precise și relevante decât cele pe care le primești.

De exemplu, dacă construiești un chatbot care trebuie să furnizeze informații factuale, ai putea dori să setezi temperatura la o valoare mai mică pentru a te asigura că răspunsurile sunt mai precise și la subiect. În schimb, dacă construiești un asistent pentru scriere creativă, ai putea dori să setezi temperatura la o valoare mai mare pentru a încuraja ieșiri mai diverse și imaginative.

Hiperparametri: Butoane și Cadrane ale Inferenței

Când lucrezi cu modele lingvistice, vei întâlni destul de des termenul "hiperparametri". În contextul inferenței (adică, atunci când folosești modelul pentru a genera răspunsuri), hiperparametrii sunt ca butoanele și cadranele pe care le poți ajusta pentru a controla comportamentul și ieșirea modelului.

Gândește-te la asta ca la ajustarea setărilor unei mașini complexe. Așa cum ai putea roti un buton pentru a controla temperatura sau comuta un întrerupător pentru a schimba modul de operare, hiperparametrii îți permit să ajustezi fin modul în care modelul lingvistic procesează și generează text.

Câțiva hiperparametri comuni pe care îi vei întâlni în timpul inferenței includ:

- **Temperatura**: După cum tocmai am menționat, acest parametru controlează gradul de aleatoriu și creativitatea textului generat. O temperatură mai mare duce

la rezultate mai diverse şi mai imprevizibile, în timp ce o temperatură mai scăzută conduce la răspunsuri mai concentrate şi mai deterministe.

- **Eşantionarea top-p (nucleus)**: Acest parametru controlează selectarea celui mai mic set de tokeni a căror probabilitate cumulativă depăşeşte un anumit prag (p). Permite obţinerea unor rezultate mai diverse, menţinând în acelaşi timp coerenţa.

- **Eşantionarea top-k**: Această tehnică selectează cei mai probabili k tokeni următori şi redistribuie masa de probabilitate între ei. Poate ajuta la prevenirea generării de către model a unor tokeni cu probabilitate scăzută sau irelevanţi.

- **Penalizările** de **frecvenţă** şi de **prezenţă**: Aceşti parametri penalizează modelul pentru repetarea prea frecventă a aceloraşi cuvinte sau fraze (penalizare de frecvenţă) sau pentru generarea unor cuvinte care nu sunt prezente în promptul de intrare (penalizare de prezenţă). Prin ajustarea acestor valori, poţi încuraja modelul să producă rezultate mai variate şi mai relevante.

- **Lungimea maximă**: Acest hiperparametru stabileşte o limită superioară pentru numărul de tokeni (cuvinte sau subcuvinte) pe care modelul îl poate genera într-un singur răspuns. Ajută la controlul verbozităţii şi al conciziei textului generat.

Pe măsură ce experimentezi cu diferite setări ale hiperparametrilor, vei descoperi că până şi ajustările mici pot avea un impact semnificativ asupra rezultatului modelului. Este ca şi cum ai ajusta fin o reţetă – un praf de sare în plus sau un timp de gătit puţin mai lung poate face toată diferenţa în preparatul final.

Cheia este să înţelegi cum afectează fiecare hiperparametru comportamentul modelului şi să găseşti echilibrul potrivit pentru sarcina ta specifică. Nu te teme să te joci cu diferite setări şi să vezi cum influenţează acestea textul generat. În timp, vei dezvolta o intuiţie pentru hiperparametrii care trebuie ajustaţi şi cum să obţii rezultatele dorite.

Prin combinarea utilizării acestor parametri cu ingineria prompturilor, generarea augmentată prin recuperare şi ajustarea fină, poţi restrânge eficient calea şi ghida

modelul de limbaj să genereze răspunsuri mai precise, relevante și valoroase pentru cazul specific de utilizare.

Modele Brute Versus Modele Ajustate pentru Instrucţiuni

Modelele brute sunt versiunile nerafinate, neantrenate ale LLM-urilor. Imaginează-ți-le ca pe o pânză proaspătă, neinfluențată încă de antrenamente specifice pentru a înțelege sau urma instrucțiuni. Ele sunt construite pe baza datelor vaste cu care au fost antrenate inițial, capabile să genereze o gamă largă de rezultate. Cu toate acestea, fără straturi suplimentare de ajustare fină bazată pe instrucțiuni, răspunsurile lor pot fi imprevizibile și necesită prompturi mai nuanțate, atent elaborate pentru a le ghida către rezultatul dorit. Lucrul cu modele brute este similar cu încercarea de a obține comunicare de la un savant-idiot care are o cantitate vastă de cunoștințe dar nu are nicio intuiție despre ce îi ceri, cu excepția cazului în care ești extrem de precis în instrucțiunile tale. Adesea se comportă precum un papagal, în sensul că, în măsura în care le faci să spună ceva inteligibil, de cele mai multe ori doar repetă ceva ce te-au auzit spunând.

Pe de altă parte, modelele ajustate pentru instrucțiuni au trecut prin runde de antrenament special concepute pentru a înțelege și urma instrucțiuni. GPT-4, Claude 3 și multe alte modele LLM populare sunt toate puternic ajustate pentru instrucțiuni. Acest antrenament implică furnizarea către model a unor exemple de instrucțiuni împreună cu rezultatele dorite, învățând efectiv modelul cum să interpreteze și să execute o gamă largă de comenzi. Ca rezultat, modelele ajustate pentru instrucțiuni pot înțelege mai ușor intenția din spatele unui prompt și pot genera răspunsuri care se aliniază îndeaproape cu așteptările utilizatorului. Acest lucru le face mai prietenoase cu utilizatorul și mai ușor de folosit, în special pentru cei care nu au timpul sau expertiza necesară pentru a se angaja în ingineria extensivă a prompturilor.

Modele Brute: Pânza Nefiltrată

Modelele brute, precum Llama 2-70B sau Yi-34B, oferă un acces mai nefiltrat la capacitățile modelului decât ceea ce ai putea fi obișnuit dacă ai experimentat cu LLM-uri populare precum GPT-4. Aceste modele nu sunt pre-ajustate pentru a urma instrucțiuni specifice, oferindu-ți o pânză goală pentru a manipula direct rezultatul modelului prin ingineria atentă a prompturilor. Această abordare necesită o înțelegere profundă a modului în care să creezi prompturi care ghidează AI-ul în direcția dorită fără a-l instrui explicit. Este similar cu a avea acces direct la straturile "brute" ale AI-ului subiacent, fără straturi intermediare care să interpreteze sau să ghideze răspunsurile modelului (de aici și numele).

Provocarea cu modelele brute constă în tendința lor de a cădea în tipare repetitive sau de a produce rezultate aleatorii. Cu toate acestea, prin ingineria atentă a prompturilor și ajustarea parametrilor precum penalizările de repetiție, modelele brute pot fi determinate să genereze conținut unic și creativ. Acest proces nu este lipsit de compromisuri; în timp ce modelele brute oferă o flexibilitate neegalată pentru inovație, ele necesită un nivel mai ridicat de expertiză.

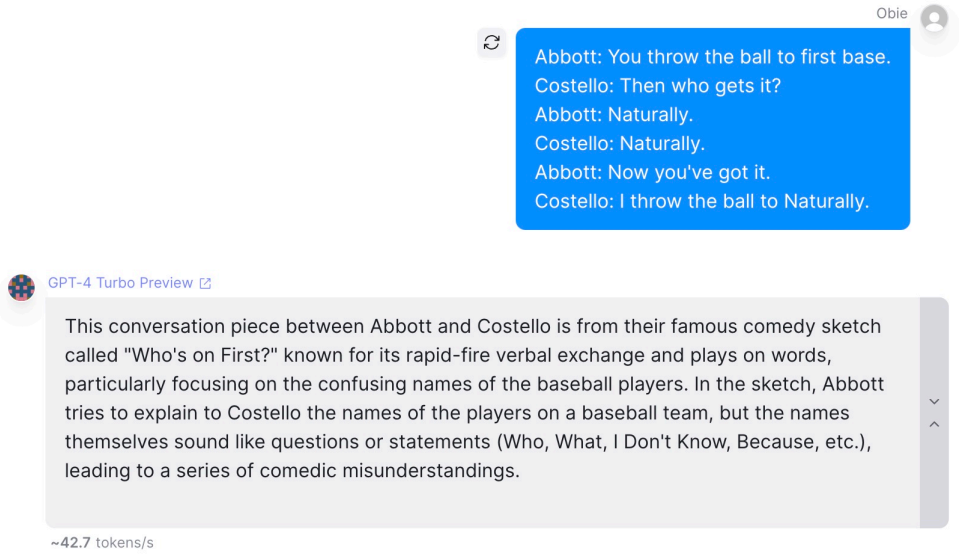


Figura 3. Pentru comparație, iată același prompt ambiguu furnizat către GPT-4

Modele Ajustate prin Instrucţiuni: Experienţa Ghidată

Modelele ajustate prin instrucțiuni sunt concepute pentru a înțelege și urma instrucțiuni specifice, făcându-le mai ușor de utilizat și accesibile pentru o gamă mai largă de aplicații. Ele înțeleg mecanica unei *conversații* și faptul că ar trebui să se oprească din generare când este *sfârșitul rândului lor de a vorbi.* Pentru mulți dezvoltatori, în special cei care lucrează la aplicații simple, modelele ajustate prin instrucțiuni oferă o soluție convenabilă și eficientă.

Procesul de ajustare prin instrucțiuni implică antrenarea modelului pe un corpus mare de prompturi și răspunsuri generate de oameni. Un exemplu notabil este setul de date open source databricks-dolly-15k, care conține peste 15.000 de perechi prompt/răspuns create de angajații Databricks pe care le puteți inspecta singuri. Setul de date acoperă opt categorii diferite de instrucțiuni, inclusiv scriere creativă, răspunsuri la întrebări închise și deschise, sumarizare, extragerea informațiilor, clasificare și brainstorming.

În timpul procesului de generare a datelor, contribuitorilor li s-au oferit îndrumări despre cum să creeze prompturi și răspunsuri pentru fiecare categorie. De exemplu, pentru sarcinile de scriere creativă, au fost instruiți să furnizeze constrângeri specifice, instrucțiuni sau cerințe pentru a ghida rezultatul modelului. Pentru răspunsurile la întrebări închise, li s-a cerut să scrie întrebări care necesită răspunsuri corecte din punct de vedere factual bazate pe un pasaj dat din Wikipedia.

Setul de date rezultat servește ca o resursă valoroasă pentru ajustarea fină a modelelor mari de limbaj pentru a prezenta capacitățile interactive și de urmărire a instrucțiunilor ale sistemelor precum ChatGPT. Prin antrenarea pe o gamă diversă de instrucțiuni și răspunsuri generate de oameni, modelul învață să înțeleagă și să urmeze directive specifice, devenind mai abil în gestionarea unei varietăți largi de sarcini.

Pe lângă ajustarea fină directă, prompturile de instrucțiuni din seturi de date precum databricks-dolly-15k pot fi utilizate și pentru generarea de date sintetice. Prin transmiterea prompturilor generate de contribuitori ca exemple cu puține iterații către un model mare de limbaj deschis, dezvoltatorii pot genera un corpus mult mai mare de instrucțiuni în fiecare categorie. Această abordare, descrisă în lucrarea Self-Instruct, permite crearea unor modele mai robuste de urmărire a instrucțiunilor.

Mai mult, instrucțiunile și răspunsurile din aceste seturi de date pot fi augmentate prin tehnici precum parafrazarea. Prin reformularea fiecărui prompt sau răspuns scurt și asocierea textului rezultat cu eșantionul de referință corespunzător, dezvoltatorii pot introduce o formă de regularizare care îmbunătățește capacitatea modelului de a urma instrucțiuni.

Ușurința în utilizare oferită de modelele instruite prin comenzi vine cu prețul unei oarecare flexibilități. Aceste modele sunt adesea puternic cenzurate, ceea ce înseamnă că nu pot oferi întotdeauna nivelul de libertate creativă necesar pentru anumite sarcini. Rezultatele lor sunt puternic influențate de prejudecățile și limitările inerente datelor lor de ajustare fină.

În ciuda acestor limitări, modelele instruite prin comenzi au devenit din ce în ce mai

populare datorită naturii lor prietenoase cu utilizatorul și capacității de a gestiona o gamă largă de sarcini cu un minim de inginerie a prompturilor. Pe măsură ce mai multe seturi de date de instrucțiuni de înaltă calitate devin disponibile, ne putem aștepta să vedem îmbunătățiri suplimentare în performanța și versatilitatea acestor modele.

Alegerea Tipului Potrivit de Model pentru Proiectul Tău

Decizia între modelele de bază (brute) și cele instruite prin comenzi depinde în cele din urmă de cerințele specifice ale proiectului tău. Pentru sarcinile care necesită un grad ridicat de creativitate și originalitate, modelele de bază oferă un instrument puternic pentru inovație. Aceste modele permit dezvoltatorilor să exploreze întregul potențial al LLM-urilor, împingând limitele a ceea ce se poate realiza prin aplicații bazate pe IA, dar necesită o abordare mai practică și o dorință de a experimenta. Temperatura și alte setări au un efect mult mai mare în modelele de bază decât în omologiile lor instruite.

Tot ce incluzi în promptul tău este ceea ce modelele de bază vor încerca să repete. Așa că, de exemplu, dacă promptul tău este o transcriere de chat, modelul brut va încerca să continue chatul. În funcție de limita maximă de tokeni, nu va genera doar următorul mesaj din chat, ci ar putea avea o întreagă conversație cu sine însuși!

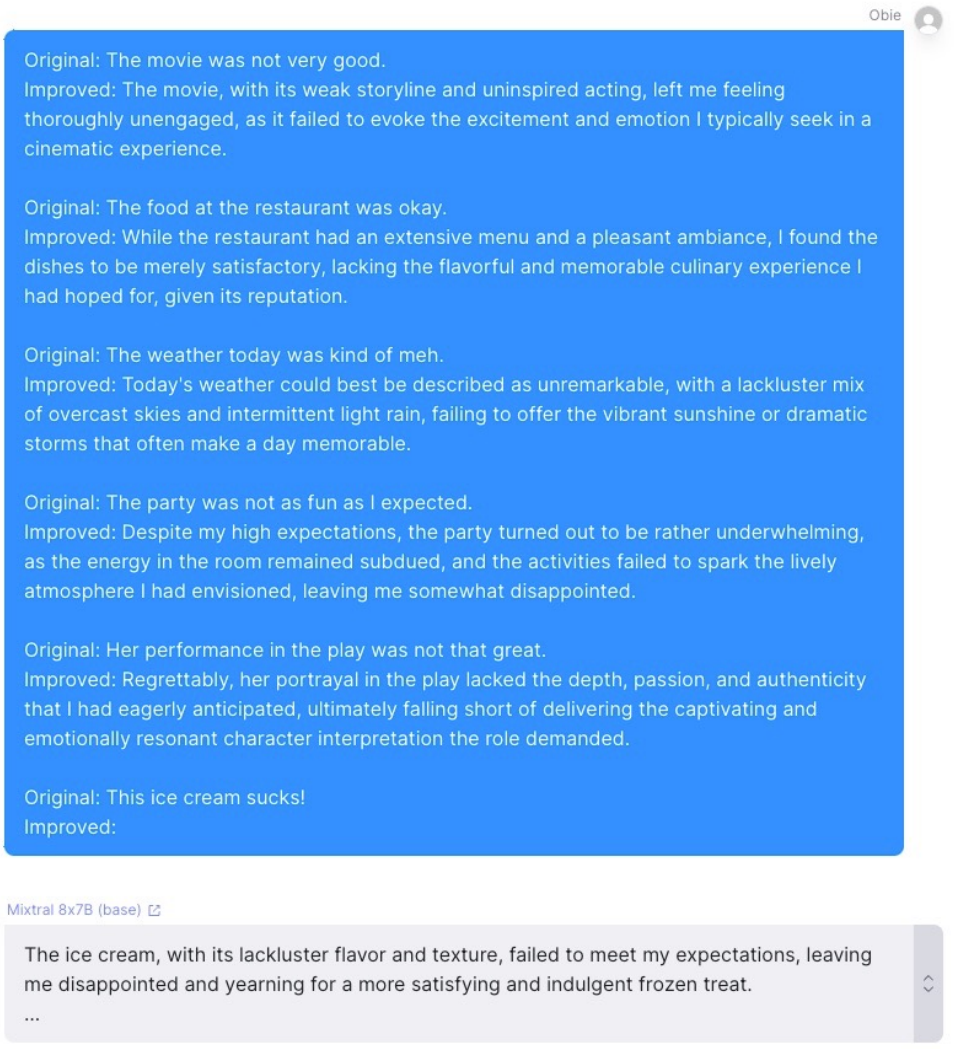


Figura 4. Exemplu de Rescriere a Propozițiilor cu Few-Shot Completion folosind Mixtral 8x7B (base)

În timp ce pregăteam exemplul de Rescriere a Propozițiilor de mai sus al utilizatorului Reddit phree_radical, am reușit să obțin rezultate utilizabile doar după multe experimente cu setările parametrilor, stabilind în cele din urmă: Temperatura 0.08, Top P: 0.2, Top K: 1 și Penalizare pentru Repetiție: 1.26.

Încercarea de a utiliza această abordare cu un model de bază în producție ar fi dificilă din cauza efectului puternic al parametrului `max_tokens`. Dacă îl setezi prea scurt, ieșirea este trunchiată. Dacă îl setezi mai lung decât are nevoie modelul pentru ieșirea dorită, va continua să halucineze exemple suplimentare.

Concluzia este că, dacă nu ai cu adevărat nevoie de control total și lipsă de cenzură, modelele instruite prin comenzi îți pot eficientiza semnificativ procesul de dezvoltare. Pentru a sublinia acest aspect, iată răspunsul Mixtral 8x7B la același prompt, dar de data aceasta în versiunea sa instruită prin comenzi:

> Îmi pare rău să vă informez că înghețata nu îndeplinește așteptările mele, deoarece îi lipsește textura bogată, cremoasă și gustul încântător pe care le asociez de obicei cu un desert de înaltă calitate. Speram la o experiență mai satisfăcătoare și mai plăcută.

În mod notabil, am putut lăsa setarea pentru numărul maxim de tokeni la 500, iar modelul s-a oprit în mod fiabil la sfârșitul rezultatului dorit fără să halucineze exemple suplimentare.

Ingineria Prompturilor

Pe măsură ce începi să aplici AI în proiectele tale, vei descoperi rapid că una dintre cele mai cruciale abilități pe care trebuie să le stăpânești este arta ingineriei prompturilor. Dar ce este exact ingineria prompturilor și de ce este atât de importantă?

În esență, ingineria prompturilor este procesul de proiectare și elaborare a prompturilor de intrare pe care le furnizezi unui model de limbaj pentru a-i ghida rezultatul. Este vorba despre înțelegerea modului de a comunica eficient cu AI-ul, folosind o combinație de instrucțiuni, exemple și context pentru a direcționa modelul către generarea răspunsului dorit.

Gândește-te la asta ca la o conversație cu un prieten foarte inteligent, dar care interpretează lucrurile destul de literal. Pentru a obține maximum din interacțiune, trebuie să fii clar, specific și să oferi suficient context pentru a te asigura că prietenul tău înțelege exact ce îi ceri. Aici intervine ingineria prompturilor și, chiar dacă la început pare ușor, crede-mă că este nevoie de multă practică pentru a o stăpâni.

Elementele de Bază ale Prompturilor Eficiente

Pentru a începe să creezi prompturi eficiente, mai întâi trebuie să înțelegi componentele cheie care alcătuiesc o intrare bine elaborată. Iată câteva dintre elementele esențiale:

1. **Instrucțiuni**: Instrucțiuni clare și concise care spun modelului ce vrei să facă. Acestea pot fi orice, de la "Rezumă următorul articol" la "Generează o poezie despre un apus" sau "transformă această cerere de modificare a proiectului într-un obiect JSON".
2. **Context**: Informații relevante care ajută modelul să înțeleagă contextul și scopul sarcinii. Acestea pot include detalii despre publicul țintă, tonul și stilul dorit sau orice constrângeri sau cerințe specifice pentru rezultat, cum ar fi o schemă JSON de respectat.
3. **Exemple**: Exemple concrete care demonstrează tipul de rezultat pe care îl cauți. Oferind câteva exemple bine alese, poți ajuta modelul să învețe tiparele și caracteristicile răspunsului dorit.
4. **Formatarea Intrării**: Separatoarele de linii și formatarea markdown oferă structură promptului nostru. Separarea promptului în paragrafe ne permite să grupăm instrucțiunile conexe, astfel încât să fie mai ușor de înțeles atât pentru oameni, cât și pentru AI. Punctele și listele numerotate ne permit să definim liste și ordinea elementelor. Marcajele pentru bold și italic ne permit să evidențiem accentul.
5. **Formatarea Ieșirii**: Instrucțiuni specifice despre cum ar trebui structurat și formatat rezultatul. Acestea pot include directive despre lungimea dorită,

utilizarea titlurilor sau a punctelor, formatarea markdown sau orice alte șabloane sau convenții specifice de ieșire care ar trebui urmate.

Prin combinarea acestor elemente de bază în diferite moduri, poți crea prompturi adaptate nevoilor tale specifice și ghida modelul către generarea unor răspunsuri relevante și de înaltă calitate.

Arta şi Ştiinţa Proiectării Prompturilor

Crearea prompturilor eficiente este atât o artă, cât și o știință. (De aceea o numim meșteșug.) Necesită o înțelegere profundă a capacităților și limitărilor modelelor de limbaj, precum și o abordare creativă în proiectarea prompturilor care să determine comportamentul dorit. Creativitatea implicată este ceea ce o face atât de distractivă, cel puțin pentru mine. Poate fi, de asemenea, foarte frustrantă, mai ales când cauți un comportament determinist

Un aspect cheie al ingineriei prompturilor este înțelegerea modului de a echilibra specificitatea și flexibilitatea. Pe de o parte, vrei să oferi suficientă îndrumare pentru a ghida modelul în direcția corectă. Pe de altă parte, nu vrei să fii atât de prescriptiv încât să limitezi capacitatea modelului de a-și utiliza propria creativitate și flexibilitate pentru a face față cazurilor limită.

O altă considerație importantă este utilizarea exemplelor. Exemplele bine alese pot fi incredibil de puternice în a ajuta modelul să înțeleagă tipul de rezultat pe care îl cauți. Cu toate acestea, este important să folosești exemplele cu judiciozitate și să te asiguri că sunt reprezentative pentru răspunsul dorit. Un exemplu prost este, în cel mai bun caz, o risipă de tokeni și, în cel mai rău caz, poate ruina rezultatul dorit.

Tehnici şi Bune Practici în Ingineria Prompturilor

Pe măsură ce te adâncești în lumea ingineriei prompturilor, vei descoperi o serie de tehnici și bune practici care te pot ajuta să creezi prompturi mai eficiente. Iată câteva domenii cheie de explorat:

1. **Învățare fără exemple vs. învățare cu puține exemple**: Înțelegerea momentului când să folosești învățarea fără exemple (fără a furniza exemple) versus învățarea cu un exemplu sau cu puține exemple (furnizând un număr mic de exemple) te poate ajuta să creezi prompturi mai eficiente și mai eficace.
2. **Rafinare iterativă**: Procesul de rafinare iterativă a prompturilor pe baza rezultatului modelului vă poate ajuta să găsiți designul optimal al promptului. Feedback Loop este o abordare puternică care folosește rezultatul propriu-zis al modelului de limbaj pentru a îmbunătăți progresiv calitatea și relevanța conținutului generat.
3. **Înlănțuirea prompturilor**: Combinarea mai multor prompturi într-o secvență vă poate ajuta să împărțiți sarcinile complexe în pași mai mici și mai ușor de gestionat. Prompt Chaining implică împărțirea unei sarcini sau conversații complexe într-o serie de prompturi mai mici, interconectate. Prin înlănțuirea prompturilor, puteți ghida AI-ul printr-un proces în mai mulți pași, menținând contextul și coerența pe parcursul interacțiunii.
4. **Ajustarea prompturilor**: Personalizarea prompturilor pentru domenii sau sarcini specifice vă poate ajuta să creați prompturi mai specializate și mai eficiente. Prompt Template vă ajută să creați structuri de prompturi flexibile, reutilizabile și ușor de întreținut, care sunt mai ușor de adaptat la sarcina respectivă.

Învățarea momentului potrivit pentru a utiliza învățarea fără exemple (zero-shot), cu un singur exemplu (one-shot) sau cu câteva exemple (few-shot) este o parte deosebit de importantă a stăpânirii ingineriei prompturilor. Fiecare abordare are propriile puncte forte și slabe, iar înțelegerea momentului când să folosiți fiecare dintre ele vă poate ajuta să creați prompturi mai eficiente și mai eficace.

Învăţarea fără exemple (Zero-Shot Learning): Când nu sunt necesare exemple

Învățarea fără exemple se referă la capacitatea unui model de limbaj de a efectua o sarcină fără exemple sau instruire explicită. Cu alte cuvinte, furnizați modelului un prompt care descrie sarcina, iar modelul generează un răspuns bazându-se exclusiv pe cunoștințele sale preexistente și înțelegerea limbajului.

Învățarea fără exemple este deosebit de utilă când:

1. Sarcina este relativ simplă și directă, iar modelul este probabil să fi întâlnit sarcini similare în timpul pre-antrenării.
2. Doriți să testați capacitățile inerente ale modelului și să vedeți cum răspunde la o sarcină nouă fără îndrumare suplimentară.
3. Lucrați cu un model de limbaj mare și divers care a fost antrenat pe o gamă largă de sarcini și domenii.

Cu toate acestea, învățarea fără exemple poate fi și imprevizibilă și nu poate produce întotdeauna rezultatele dorite. Răspunsul modelului poate fi influențat de prejudecăți sau inconsecvențe în datele de pre-antrenare și poate avea dificultăți cu sarcini mai complexe sau nuanțate.

> Am văzut prompturi fără exemple care funcționează bine pentru 80% din cazurile mele de test și produc rezultate complet greșite sau de neînțeles pentru celelalte 20%. Este foarte important să implementați un regim de testare riguros, mai ales dacă vă bazați mult pe prompturi fără exemple.

Învăţarea cu un singur exemplu (One-Shot Learning): Când un singur exemplu poate face diferenţa

Învățarea cu un singur exemplu implică furnizarea către model a unui singur exemplu al rezultatului dorit împreună cu descrierea sarcinii. Acest exemplu serveşte ca şablon sau model pe care modelul îl poate utiliza pentru a genera propriul răspuns.

Învățarea cu un singur exemplu poate fi eficientă când:

1. Sarcina este relativ nouă sau specifică, iar modelul este posibil să nu fi întâlnit multe exemple similare în timpul pre-antrenării.
2. Doriți să oferiți o demonstrație clară și concisă a formatului sau stilului dorit pentru rezultat.
3. Sarcina necesită o structură sau o convenție specifică care poate să nu fie evidentă doar din descrierea sarcinii.

Descrierile care sunt evidente pentru dumneavoastră nu sunt neapărat evidente pentru AI. Exemplele cu un singur caz pot ajuta la clarificarea lucrurilor.

Învățarea cu un singur exemplu poate ajuta modelul să înțeleagă mai clar așteptările și să genereze un răspuns care este mai bine aliniat cu exemplul furnizat. Cu toate acestea, este important să alegeți exemplul cu atenție și să vă asigurați că este reprezentativ pentru rezultatul dorit. Când alegeți exemplul, întrebați-vă despre potențialele cazuri limită și gama de date de intrare pe care le va gestiona promptul.

Figura 5. Un exemplu cu un singur caz al JSON-ului dorit

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

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

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

This is an example of well-formed output:

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

Few-Shot Learning: Când Mai Multe Exemple Pot Îmbunătăţi Performanţa

Few-shot learning implică furnizarea către model a unui număr mic de exemple (de obicei între 2 și 10) împreună cu descrierea sarcinii. Aceste exemple servesc la oferirea mai multor contexte și variații modelului, ajutându-l să genereze răspunsuri mai diverse și mai precise.

Few-shot learning este deosebit de util când:

1. Sarcina este complexă sau nuanțată, iar un singur exemplu poate să nu fie suficient pentru a capta toate aspectele relevante.
2. Doriți să oferiți modelului o serie de exemple care demonstrează diferite variații sau cazuri limită.

3. Sarcina necesită ca modelul să genereze răspunsuri care sunt în concordanţă cu un domeniu sau stil specific.

Prin furnizarea mai multor exemple, puteţi ajuta modelul să dezvolte o înţelegere mai robustă a sarcinii şi să genereze răspunsuri mai consistente şi mai fiabile.

Exemplu: Prompturile Pot Fi Mult Mai Complexe Decât Vă Imaginaţi

Modelele lingvistice mari de astăzi sunt mult mai puternice şi capabile de raţionament decât v-aţi putea imagina. Aşadar, nu vă limitaţi la a gândi prompturile ca fiind doar o specificaţie de perechi input-output. Puteţi experimenta oferind instrucţiuni lungi şi complexe în moduri care amintesc de interacţiunea cu un om.

De exemplu, acesta este un prompt pe care l-am folosit în Olympia când făceam prototipul integrării noastre cu serviciile Google, care în totalitatea sa este probabil unul dintre cele mai mari API-uri din lume. Experimentele mele anterioare au dovedit că GPT-4 are o cunoaştere decentă a API-ului Google, şi nu aveam timp sau motivaţie să scriu un strat de mapare granular, implementând fiecare funcţie pe care voiam să o ofer AI-ului meu una câte una. Ce-ar fi dacă aş putea să ofer AI-ului acces la *tot* API-ul Google?

Mi-am început promptul spunând AI-ului că are acces direct la punctele finale ale API-ului Google prin HTTP şi că rolul său este să utilizeze aplicaţiile şi serviciile Google în numele utilizatorului. Apoi am oferit îndrumări, reguli legate de parametrul `fields`, deoarece părea să aibă cele mai multe probleme cu acesta, şi câteva indicii specifice API-ului (few-shot prompting, în acţiune).

Iată întregul prompt, care spune AI-ului cum să utilizeze funcţia `invoke_google_api` furnizată.

```
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
```

Poate te întrebi dacă acest prompt funcționează. Răspunsul simplu este da. AI-ul nu a ştiut întotdeauna să apeleze API-ul perfect din prima încercare. Cu toate acestea, dacă făcea o greşeală, pur şi simplu transmiteam mesajele de eroare rezultate înapoi ca rezultat al apelului. Cunoscând eroarea sa, AI-ul putea să analizeze greşeala şi să încerce din nou. În majoritatea cazurilor, reuşea să o facă corect după câteva încercări.

Să ştii că structurile JSON mari pe care API-ul Google le returnează ca date în timp ce folosește acest prompt sunt extrem de ineficiente, așa că *nu* recomand să folosești această abordare în producție. Cu toate acestea, cred că faptul că această abordare a funcționat este o dovadă a puterii pe care o poate avea ingineria prompturilor.

Experimentare şi Iteraţie

În cele din urmă, modul în care îți proiectezi promptul depinde de sarcina specifică, de complexitatea rezultatului dorit și de capacitățile modelului lingvistic cu care lucrezi.

Ca inginer de prompturi, este important să experimentezi cu diferite abordări și să iterezi pe baza rezultatelor. Începe cu învățarea fără exemple și vezi cum se comportă modelul. Dacă rezultatul este inconsistent sau nesatisfăcător, încearcă să oferi unul sau mai multe exemple și vezi dacă performanța se îmbunătățește.

Ține cont că chiar și în cadrul fiecărei abordări, există loc pentru variație și optimizare. Poți experimenta cu exemple diferite, poți ajusta formularea descrierii sarcinii sau poți oferi context suplimentar pentru a ajuta la ghidarea răspunsului modelului.

În timp, vei dezvolta o intuiție pentru ce abordare are șanse mai mari să funcționeze cel mai bine pentru o anumită sarcină și vei putea să creezi prompturi mai eficiente și mai eficace. Cheia este să rămâi curios, experimental și iterativ în abordarea ta față de ingineria prompturilor.

Pe parcursul acestei cărți, vom explora mai în profunzime aceste tehnici și vom analiza cum pot fi aplicate în scenarii din lumea reală. Stăpânind arta și știința ingineriei prompturilor, vei fi bine pregătit pentru a debloca întregul potențial al dezvoltării aplicațiilor bazate pe AI.

Arta Vagului

Când vine vorba de crearea unor prompturi eficiente pentru modelele lingvistice mari (MLM), o presupunere comună este că mai multă specificitate și instrucțiuni detaliate duc la rezultate mai bune. Cu toate acestea, experiența practică a arătat că acest lucru nu este întotdeauna adevărat. De fapt, a fi intenționat vag în prompturile tale poate adesea să producă rezultate superioare, folosind capacitatea remarcabilă a MLM de a generaliza și a face deducții.

Ken, un fondator de startup care a procesat peste 500 de milioane de tokeni GPT, a împărtășit perspective valoroase din experiența sa. Una dintre lecțiile cheie pe care le-a învățat a fost că "mai puțin înseamnă mai mult" când vine vorba de prompturi. În loc de liste exacte sau instrucțiuni excesiv de detaliate, Ken a descoperit că permiterea MLM-ului să se bazeze pe cunoștințele sale de bază a produs adesea rezultate mai bune.

Această realizare răstoarnă mentalitatea tradițională a programării explicite, unde totul trebuie să fie explicat în detaliu meticuloș. Cu MLM-urile, este important să recunoaștem că acestea posedă o cantitate vastă de cunoștințe și pot face conexiuni și deducții inteligente. Fiind mai vag în prompturile tale, oferi MLM-ului libertatea de a-și folosi înțelegerea și de a veni cu soluții pe care s-ar putea să nu le fi specificat explicit.

De exemplu, când echipa lui Ken lucra la un pipeline pentru a clasifica textul ca fiind legat de unul dintre cele 50 de state ale SUA sau de guvernul federal, abordarea lor inițială implica furnizarea unei liste *complete* și detaliate a statelor și ID-urilor lor corespunzătoare sub forma unui array formatat JSON.

```
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} ... ]
```

Abordarea a eșuat într-o măsură care i-a forțat să analizeze mai profund prompt-ul pentru a găsi modalități de îmbunătățire. În acest proces, au observat că deși MLM-ul deseori greșea id-ul, returna în mod constant numele complet al statului corect într-un câmp `name`, *chiar dacă nu ceruseră explicit acest lucru.*

Prin eliminarea id-urilor de localitate și simplificarea prompt-ului la ceva de genul "Tu evident cunoști cele 50 de state, GPT, așa că oferă-mi doar numele complet al statului la care se referă acest lucru, sau Federal dacă se referă la guvernul SUA", au obținut rezultate mai bune. Această experiență subliniază puterea de a valorifica capacitățile de generalizare ale MLM-ului și de a-i permite să facă deducții bazate pe cunoștințele sale existente.

> Justificarea lui Ken pentru această abordare particulară de clasificare, în comparație cu o tehnică de programare mai tradițională, ilustrează mentalitatea celor dintre noi care am îmbrățișat potențialul tehnologiei MLM: "Aceasta nu este o sarcină dificilă – probabil am fi putut folosi string/regex, dar există suficiente cazuri particulare ciudate încât ar fi durat mai mult."

Capacitatea MLM-urilor de a îmbunătăți calitatea și generalizarea atunci când li se oferă prompt-uri mai vagi este o caracteristică remarcabilă a gândirii și delegării de ordin superior. Aceasta demonstrează că MLM-urile pot gestiona ambiguitatea și pot lua decizii inteligente bazate pe contextul furnizat.

Cu toate acestea, este important de menționat că a fi vag nu înseamnă a fi neclar sau ambiguu. Cheia este să oferi suficient context și îndrumare pentru a ghida MLM-ul în direcția corectă, permițându-i în același timp flexibilitatea de a-și utiliza cunoștințele și capacitățile de generalizare.

Prin urmare, când concepi prompt-uri, ia în considerare următoarele sfaturi de tipul "mai puțin înseamnă mai mult":

1. Concentrează-te pe rezultatul dorit în loc să specifici fiecare detaliu al procesului.
2. Oferă context și constrângeri relevante, dar evită supraspecificarea.
3. Valorifică cunoștințele existente făcând referire la concepte sau entități comune.
4. Lasă loc pentru deducții și conexiuni bazate pe contextul dat.
5. Iterează și rafinează prompt-urile pe baza răspunsurilor MLM-ului, găsind echilibrul potrivit între specificitate și vagitate.

Prin îmbrățișarea artei vagității în ingineria prompt-urilor, poți debloca întregul potențial al MLM-urilor și obține rezultate mai bune. Ai încredere în capacitatea MLM-ului de a generaliza și de a lua decizii inteligente, și s-ar putea să fii surprins de calitatea și creativitatea rezultatelor pe care le primești. Acordă atenție modului în

care diferitele modele răspund la diferite niveluri de specificitate în prompt-urile tale și ajustează în consecință. Cu practică și experiență, vei dezvolta un simț ascuțit pentru când să fii mai vag și când să oferi îndrumare suplimentară, permițându-ți să valorifici eficient puterea MLM-urilor în aplicațiile tale.

De ce antropomorfismul domină ingineria prompt-urilor

Antropomorfismul, atribuirea caracteristicilor umane entităților non-umane, este abordarea dominantă în ingineria prompt-urilor pentru modele lingvistice mari din motive deliberate. Este o alegere de design care face interacțiunea cu sisteme AI puternice mai intuitivă și accesibilă pentru o gamă largă de utilizatori (inclusiv pentru noi, dezvoltatorii de aplicații).

Antropomorfizarea MLM-urilor oferă un cadru care este imediat intuitiv pentru persoanele care sunt complet nefamiliarizate cu complexitățile tehnice subiacente ale sistemului. După cum vei experimenta dacă încerci să folosești un model care nu este instruct-tuned pentru a face ceva util, construirea unui cadru în care continuarea așteptată oferă valoare este o sarcină provocatoare. Necesită o înțelegere destul de profundă a funcționării interne a sistemului, ceva ce posedă un număr relativ mic de experți.

Prin tratarea interacțiunii cu un model lingvistic ca pe o conversație între două persoane, ne putem baza pe înțelegerea noastră înnăscută a comunicării umane pentru a ne transmite nevoile și așteptările. La fel cum designul UI-ului primelor Macintosh a prioritizat intuitivitatea imediată în detrimentul sofisticării, încadrarea antropomorfică a AI-ului ne permite să interacționăm într-un mod care se simte natural și familiar.

Când comunicăm cu o altă persoană, instinctul nostru este să ne adresăm direct folosind "tu" și să oferim instrucțiuni clare despre cum ne așteptăm să se comporte. Acest lucru se traduce perfect în procesul de inginerie a prompt-urilor, unde ghidăm comportamentul AI-ului specificând prompt-uri de sistem și angajându-ne într-un dialog de tip du-te-vino.

Prin încadrarea interacţiunii în acest mod, putem înţelege uşor conceptul de a oferi instrucţiuni AI-ului şi de a primi răspunsuri relevante în schimb. Abordarea antropomorfică reduce încărcătura cognitivă şi ne permite să ne concentrăm pe sarcina la îndemână în loc să ne luptăm cu complexităţile tehnice ale sistemului.

Este important de menţionat că, deşi antropomorfismul este un instrument puternic pentru a face sistemele AI mai accesibile, acesta vine şi cu anumite riscuri şi limitări. Utilizatorul nostru poate dezvolta aşteptări nerealiste sau poate forma ataşamente emoţionale nesănătoase faţă de sistemele noastre. Ca ingineri de prompt-uri şi dezvoltatori, este crucial să găsim un echilibru între valorificarea beneficiilor antropomorfismului şi asigurarea că utilizatorii menţin o înţelegere clară a capacităţilor şi limitărilor AI-ului.

Pe măsură ce domeniul ingineriei prompturilor continuă să evolueze, ne putem aştepta să vedem rafinări şi inovaţii suplimentare în modul în care interacţionăm cu modelele lingvistice mari. Cu toate acestea, antropomorfismul ca mijloc de a oferi o experienţă intuitivă şi accesibilă pentru dezvoltatori şi utilizatori va rămâne probabil un principiu fundamental în proiectarea acestor sisteme.

Separarea Instrucţiunilor de Date: Un Principiu Crucial

Este esenţial să înţelegem un principiu fundamental care stă la baza securităţii şi fiabilităţii acestor sisteme: separarea instrucţiunilor de date.

În informatică, distincţia clară între datele pasive şi instrucţiunile active este un principiu de bază al securităţii. Această separare ajută la prevenirea executării neintenţionate sau maliţioase a codului care ar putea compromite integritatea şi stabilitatea sistemului. Cu toate acestea, LLM-urile de astăzi, care au fost dezvoltate în principal ca modele care urmează instrucţiuni, precum chatboţii, adesea duc lipsă de această separare formală şi principială.

În ceea ce priveşte LLM-urile, instrucţiunile pot apărea oriunde în input, fie că este vorba de un prompt de sistem sau un prompt furnizat de utilizator. Această lipsă de

separare poate duce la vulnerabilități potențiale și comportamente nedorite, similare cu problemele întâlnite de bazele de date cu injecții SQL sau sistemele de operare fără protecție adecvată a memoriei.

În timp ce lucrați cu LLM-uri, este crucial să fiți conștienți de această limitare și să luați măsuri pentru a atenua riscurile. O abordare este să vă construiți cu atenție prompturile și input-urile pentru a face o distincție clară între instrucțiuni și date. Metodele tipice pentru oferirea de îndrumări explicite despre ce constituie o instrucțiune și ce ar trebui tratat ca date pasive implică marcarea de tip markup. Promptul dumneavoastră poate ajuta LLM-ul să înțeleagă și să respecte mai bine această separare.

Figura 6. Utilizarea XML pentru a face distincția între instrucțiuni, materialul sursă și promptul utilizatorului

```
<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>
```

O altă tehnică este implementarea unor straturi suplimentare de validare și sanitizare a datelor de intrare furnizate modelului LLM. Prin filtrarea sau escaparea oricăror potențiale instrucțiuni sau fragmente de cod care ar putea fi încorporate în date, puteți reduce șansele unei executări nedorite. Modele precum Înlănțuirea Prompturilor sunt utile în acest scop.

Mai mult, în timp ce vă proiectați arhitectura aplicației, luați în considerare încorporarea unor mecanisme care să impună separarea instrucțiunilor și a datelor la un nivel superior. Aceasta ar putea implica utilizarea unor endpoint-uri sau API-uri separate pentru gestionarea instrucțiunilor și datelor, implementarea unei validări și analize stricte a datelor de intrare și aplicarea *principiului privilegiului minim* pentru a limita domeniul de acțiune a ceea ce LLM-ul poate accesa și executa.

Principiul Privilegiului Minim

Adoptarea principiului privilegiului minim este ca și cum ai organiza o petrecere foarte exclusivistă unde invitații primesc acces doar în camerele în care au absolută nevoie să fie. Imaginează-ți că găzduiești această petrecere într-o vilă impunătoare. Nu toată lumea are nevoie să hoinărească prin pivnița cu vinuri sau dormitorul principal, nu-i așa? Aplicând acest principiu, practic oferi chei care deschid doar anumite uși, asigurându-te că fiecare invitat sau, în cazul nostru, fiecare componentă a aplicației tale LLM, are doar accesul necesar pentru a-și îndeplini rolul.

Nu este vorba doar despre a fi zgârcit cu cheile, ci despre recunoașterea faptului că într-o lume în care amenințările pot veni de oriunde, strategia inteligentă este să limitezi terenul de joacă. Dacă cineva nechemat reușește să se strecoare la petrecerea ta, se va trezi blocat în hol, ca să spunem așa, limitând dramatic răul pe care îl poate face. Așadar, când securizezi aplicațiile LLM, ține minte: oferă chei doar pentru camerele care sunt necesare și păstrează restul vilei în siguranță. Nu este doar o chestiune de bună-cuviință; este o măsură bună de securitate.

În timp ce starea actuală a LLM-urilor poate să nu aibă o separare formală între instrucțiuni și date, este esențial pentru tine, ca dezvoltator, să fii conștient de această limitare și să iei măsuri proactive pentru a atenua riscurile. Prin aplicarea celor mai bune practici din informatică și adaptarea acestora la caracteristicile unice ale LLM-

urilor, poți construi aplicații mai sigure și mai fiabile care valorifică puterea acestor modele, menținând în același timp integritatea sistemului tău.

Distilarea Prompturilor

Crearea promptului perfect este adesea o sarcină dificilă și consumatoare de timp, necesitând o înțelegere profundă a domeniului țintă și a nuanțelor modelelor de limbaj. Aici intervine tehnica "Distilării Prompturilor", oferind o abordare puternică în ingineria prompturilor care valorifică capabilitățile modelelor de limbaj de mari dimensiuni (LLM) pentru a eficientiza și optimiza procesul.

Distilarea Prompturilor este o tehnică în mai multe etape care implică utilizarea LLM-urilor pentru a asista în crearea, rafinarea și optimizarea prompturilor. În loc să se bazeze exclusiv pe expertiza și intuiția umană, această abordare folosește cunoștințele și capacitățile generative ale LLM-urilor pentru a crea prompt-uri de înaltă calitate în mod colaborativ.

Prin angajarea într-un proces iterativ de generare, rafinare și integrare, Distilarea Prompturilor îți permite să creezi prompturi mai coerente, cuprinzătoare și aliniate cu sarcina sau rezultatul dorit. Reține că procesul de distilare poate fi făcut manual în unul din numeroasele "medii de testare" furnizate de marii furnizori de AI precum OpenAI sau Anthropic, sau poate fi automatizat ca parte a codului aplicației tale, în funcție de cazul de utilizare.

Cum Funcționează

Distilarea Prompturilor implică de obicei următorii pași:

1. **Identificarea Intenției Fundamentale**: Analizează promptul pentru a determina scopul său principal și rezultatul dorit. Elimină orice informație suplimentară și concentrează-te pe intenția fundamentală a promptului.

2. **Eliminarea Ambiguității**: Revizuiește promptul pentru orice limbaj ambiguu sau vag. Clarifică sensul și oferă detalii specifice pentru a ghida AI-ul către generarea unor răspunsuri precise și relevante.
3. **Simplificarea Limbajului**: Simplifică promptul folosind un limbaj clar și concis. Evită structurile complexe ale propozițiilor, jargonul sau detaliile inutile care ar putea confunda AI-ul sau introduce zgomot.
4. **Furnizarea Contextului Relevant**: Include doar informațiile contextuale cele mai relevante necesare pentru ca AI-ul să înțeleagă și să proceseze eficient promptul. Evită includerea detaliilor irelevante sau redundante care ar putea distrage de la intenția fundamentală.
5. **Iterare și Rafinare**: Iterează și rafinează continuu promptul pe baza răspunsurilor și feedback-ului AI-ului. Evaluează rezultatele generate și fă ajustările necesare pentru a îmbunătăți claritatea și eficacitatea promptului. Opțional, versionează prompturile în baza de date folosind Obiectul Prompt pentru a ține evidența iterațiilor și a-ți oferi posibilitatea de a reveni ușor la versiuni anterioare în timpul execuției.

Generarea Promptului Inițial

Primul pas în Distilarea Prompturilor este să furnizăm LLM-ului o descriere la nivel înalt sau intenția sarcinii ori rezultatului dorit. Aceasta poate fi o afirmație succintă sau un set de cuvinte-cheie care surprind esența a ceea ce ar trebui să realizeze promptul. LLM-ul generează apoi un prompt inițial bazat pe această informație, încorporând instrucțiuni relevante, exemple sau context pe care le consideră adecvate.

Să încercăm să-i dăm lui Claude 3 Opus următorul prompt inițial:

```
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 răspunde cu următoarea ieşire:

```
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.
```

Rafinarea Promptului

Odată ce promptul inițial este generat, revizuiți-l și apoi cereți LLM-ului să ia în considerare orice context, constrângeri sau feedback suplimentar. Lăsați LLM-ul să genereze o versiune rafinată a promptului, luând în considerare inputul furnizat și propria sa înțelegere a sarcinii. Acest proces de rafinare poate fi repetat iterativ, fiecare iterație rafinând în continuare promptul pe baza colaborării dintre utilizator și LLM.

Iată solicitarea mea de rafinare, continuând procesul anterior de distilare.

```
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.
```

Și răspunsul lui Claude:

```
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.
```

Comprimarea Prompturilor

După cum probabil ai observat, LLM-urile tind să fie prolixe, cu excepția cazului în care le ceri să fie concise. Promptul la care am lucrat în secțiunile anterioare este destul de lung, așa că hai să îl punem pe Claude să îl reducă la dimensiuni mai mici:

```
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 răspunde, cu următorul prompt dramatic mai scurt:

```
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.
```

Directiva de Sistem şi Integrarea Contextului

Pe lângă rafinarea promptului în sine, LLM-ul poate genera, de asemenea, directive de sistem sau informaţii contextuale adecvate pentru a ghida rezultatul final. Când realizezi ingineria prompturilor pentru rutinele AI care vor fi integrate în codul aplicației tale, te vei concentra aproape sigur pe constrângerile de ieșire în această etapă a distilării, dar poți lucra și la tonul dorit, stil, format sau orice alți parametri relevanți care influențează răspunsul generat.

Asamblarea Promptului Final

Punctul culminant al procesului de Distilare a Prompturilor este asamblarea promptului final. Aceasta implică combinarea promptului rafinat, a directivelor de sistem generate și a contextului integrat într-un cod coerent și cuprinzător care este gata să fie utilizat pentru generarea rezultatului dorit.

Poți experimenta din nou cu compresia prompturilor în etapa de asamblare finală a promptului, cerând LLM-ului să reducă formularea promptului la cea mai scurtă serie de tokeni posibilă, păstrând în același timp esența comportamentului său. Este cu siguranță un exercițiu de tip hit or miss, dar mai ales în cazul prompturilor care vor fi rulate la scară mare, câștigurile de eficiență îți pot economisi destul de mulți bani în consumul de tokeni.

Beneficii Cheie

Prin valorificarea cunoștințelor și capacităților generative ale LLM-urilor pentru a rafina prompturile tale, prompturile rezultate sunt mai susceptibile să fie bine structurate, informative și adaptate sarcinii specifice. Procesul de rafinare iterativă ajută la asigurarea calității prompturilor și la captarea eficientă a intenției dorite. Alte beneficii includ:

Eficiență și Viteză: Distilarea Prompturilor eficientizează procesul de inginerie a prompturilor prin automatizarea anumitor aspecte ale creării și rafinării prompturilor. Natura colaborativă a tehnicii permite o convergență mai rapidă către un prompt eficient, reducând timpul și efortul necesar pentru crearea manuală a prompturilor.

Consistență și Scalabilitate: Utilizarea LLM-urilor în procesul de inginerie a prompturilor ajută la menținerea consistenței între prompturi, deoarece LLM-urile pot învăța și aplica cele mai bune practici și tipare din prompturile anterioare de succes. Această consistență, combinată cu capacitatea de a genera prompturi la scară, face

din Distilarea Prompturilor o tehnică valoroasă pentru aplicațiile bazate pe AI la scară largă.

Idee de Proiect: Instrumente la nivel de bibliotecă care simplifică procesul de versionare și evaluare a prompturilor în sisteme care efectuează distilări automate ale prompturilor ca parte a codului aplicației.

Pentru a implementa Distilarea Prompturilor, dezvoltatorii pot proiecta un flux de lucru sau o conductă care integrează LLM-uri în diverse etape ale procesului de inginerie a prompturilor. Acest lucru poate fi realizat prin apeluri API, instrumente personalizate sau medii de dezvoltare integrate care facilitează interacțiunea fără probleme între utilizatori și LLM-uri în timpul creării prompturilor. Detaliile specifice de implementare pot varia în funcție de platforma LLM aleasă și de cerințele aplicației.

Dar despre fine-tuning?

În această carte, acoperim pe larg ingineria prompturilor și RAG, dar nu și fine-tuning-ul. Motivul principal pentru această decizie este că, în opinia mea, majoritatea dezvoltatorilor de aplicații nu au nevoie de fine-tuning pentru nevoile lor de integrare AI.

Ingineria prompturilor, care implică crearea atentă a prompturilor cu exemple zero până la puține demonstrații, constrângeri și instrucțiuni, poate ghida eficient modelul pentru a genera răspunsuri relevante și precise pentru o gamă largă de sarcini. Prin furnizarea unui context clar și îngustarea căii prin prompturi bine concepute, poți valorifica vastele cunoștințe ale modelelor de limbaj mari fără a avea nevoie de fine-tuning.

În mod similar, Generarea Augmentată prin Recuperare (RAG) oferă o abordare puternică pentru integrarea AI în aplicații. Prin recuperarea dinamică a informațiilor relevante din baze de cunoștințe externe sau documente, RAG oferă modelului un context focalizat în momentul promptării. Acest lucru permite modelului să genereze

răspunsuri mai precise, actualizate și specifice domeniului, fără a necesita procesul intensiv în timp și resurse al fine-tuning-ului.

În timp ce fine-tuning-ul poate fi benefic pentru domenii foarte specializate sau sarcini care necesită un nivel profund de personalizare, acesta vine adesea cu costuri computaționale semnificative, cerințe de date și overhead de întreținere. Pentru majoritatea scenariilor de dezvoltare a aplicațiilor, combinația dintre ingineria eficientă a prompturilor și RAG ar trebui să fie suficientă pentru a obține funcționalitatea și experiența utilizatorului dorite bazate pe AI.

Generare Augmentată prin Recuperare (RAG)

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Ce este Generarea Augmentată prin Recuperare?

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Cum funcţionează RAG?

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

De ce să foloseşti RAG în aplicaţiile tale?

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Implementarea RAG în Aplicaţia Ta

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Pregătirea Surselor de Cunoştinţe (Segmentare)

Fragmentarea în Propoziţii

Note de Implementare

Verificarea Calităţii

Beneficiile Regăsirii Bazate pe Propoziţii

Exemple din Lumea Reală ale RAG

Studiu de Caz: RAG într-o Aplicație de Pregătire a Taxelor Fără Încorporări

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Optimizarea Inteligentă a Interogărilor (IQO)

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Reclasificarea

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Evaluarea RAG (RAGAs)

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Fidelitate

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Relevanța Răspunsului

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Precizia Contextului

Relevanța Contextului

Rata de Recuperare a Contextului

Rata de Recuperare a Entităților din Context

Similaritatea Semantică a Răspunsului (ANSS)

Corectitudinea Răspunsului

Critica pe Aspecte

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Provocări şi Perspective de Viitor

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Segmentarea Semantică: Îmbunătăţirea Recuperării cu Segmentare Sensibilă la Context

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Indexarea ierarhică: Structurarea datelor pentru o regăsire îmbunătăţită

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Self-RAG: O îmbunătăţire auto-reflexivă

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

HyDE: Încorporări de Documente Ipotetice

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Ce este învăţarea contrastivă?

Multitudinea de Lucrători

Îmi place să mă gândesc la componentele mele AI ca la niște mici "lucrători" virtuali, aproape umani, care pot fi integrați fără probleme în logica aplicației mele pentru a efectua sarcini specifice sau a lua decizii complexe. Ideea este de a umaniza în mod intenționat capacitățile LLM-ului, astfel încât nimeni să nu se entuziasmeze *prea* tare și să le atribuie calități magice pe care nu le posedă.

În loc să se bazeze exclusiv pe algoritmi complecși sau implementări manuale care consumă mult timp, dezvoltatorii pot conceptualiza componentele AI ca entități inteligente, dedicate, asemănătoare oamenilor, care pot fi invocate oricând este nevoie pentru a aborda probleme complexe și a oferi soluții bazate pe pregătirea și cunoștințele lor. Aceste entități nu se distrag, nu își iau concediu medical. Nu decid spontan să facă lucrurile în moduri diferite față de cum au fost instruite să le facă și, în general vorbind, dacă sunt programate corect, nici nu fac greșeli.

În termeni tehnici, principiul cheie din spatele acestei abordări este descompunerea sarcinilor complexe sau a proceselor decizionale în unități mai mici și mai ușor de gestionat, care pot fi gestionate de lucrători AI specializați. Fiecare lucrător este proiectat să se concentreze pe un aspect specific al problemei, aducând expertiza și capacitățile sale unice. Prin distribuirea volumului de muncă între mai mulți lucrători AI, aplicația poate atinge o eficiență, scalabilitate și adaptabilitate mai mare.

De exemplu, să luăm în considerare o aplicație web care necesită moderarea în timp real a conținutului generat de utilizatori. Implementarea unui sistem complet de moderare de la zero ar fi o sarcină descurajantă, necesitând un efort semnificativ de dezvoltare și întreținere continuă. Cu toate acestea, prin utilizarea abordării Multitudinii de Lucrători, dezvoltatorii pot integra lucrători de moderare bazați pe AI în logica aplicației. Acești lucrători pot analiza și marca automat conținutul inadecvat, permițând dezvoltatorilor să se concentreze pe alte aspecte critice ale aplicației.

Lucrătorii AI Ca Componente Independente Reutilizabile

Un aspect cheie al abordării Multitudinii de Lucrători este modularitatea sa. Susținătorii programării orientate pe obiecte ne spun de decenii să ne gândim la interacțiunile dintre obiecte ca la mesaje. Ei bine, lucrătorii AI pot fi proiectați ca componente independente, reutilizabile care pot "vorbi între ele" prin mesaje în limbaj simplu, aproape ca și cum ar fi într-adevăr mici oameni care vorbesc între ei. Această abordare cu cuplare slabă permite aplicației să se adapteze și să evolueze în timp, pe măsură ce apar noi tehnologii AI sau cerințele logicii de afaceri se schimbă.

În practică, necesitatea de a proiecta interfețe clare și protocoale de comunicare între componente nu s-a schimbat doar pentru că sunt implicați lucrători AI. Trebuie să iei în considerare în continuare și alți factori precum performanța, scalabilitatea și securitatea, dar acum există și cerințe complet noi "mai puțin tehnice" de luat în considerare. De

exemplu, mulți utilizatori se opun utilizării datelor lor private pentru antrenarea noilor modele AI. Ai verificat nivelul de confidențialitate oferit de furnizorul de model pe care îl utilizezi?

Lucrători AI Ca Microservicii?

Pe măsură ce citești despre abordarea Multitudinii de Lucrători, ai putea observa unele asemănări cu arhitectura de microservicii. Ambele subliniază descompunerea sistemelor complexe în unități mai mici, mai ușor de gestionat și care pot fi implementate independent. La fel cum microserviciile sunt proiectate să fie slab cuplate, concentrate pe capacități specifice de business și să comunice prin API-uri bine definite, lucrătorii AI sunt proiectați să fie modulari, specializați în sarcinile lor și să interacționeze între ei prin interfețe și protocoale de comunicare clare.

Cu toate acestea, există câteva diferențe cheie de avut în vedere. În timp ce microserviciile sunt de obicei implementate ca procese sau servicii separate care rulează pe mașini sau containere diferite, lucrătorii AI pot fi implementați ca componente de sine stătătoare în cadrul unei singure aplicații sau ca servicii separate, în funcție de cerințele specifice și nevoile de scalabilitate. În plus, comunicarea dintre lucrătorii AI implică adesea schimbul de informații bogate, bazate pe limbaj natural, cum ar fi prompturi, instrucțiuni și conținut generat, mai degrabă decât formatele de date mai structurate utilizate în mod obișnuit în microservicii.

În ciuda acestor diferențe, principiile modularității, cuplării slabe și interfețelor clare de comunicare rămân centrale pentru ambele modele. Prin aplicarea acestor principii la arhitectura lucrătorilor tăi AI, poți crea sisteme flexibile, scalabile și ușor de întreținut care valorifică puterea AI pentru a rezolva probleme complexe și a oferi valoare utilizatorilor tăi.

Abordarea Multitudinii de Lucrători poate fi aplicată în diverse domenii și aplicații, valorificând puterea AI pentru a aborda sarcini complexe și a oferi soluții inteligente. Să explorăm câteva exemple concrete despre cum lucrătorii AI pot fi folosiți în diferite contexte.

Gestionarea Conturilor

Practic, fiecare aplicație web independentă are conceptul de cont (sau utilizator). În Olympia, folosim un lucrător AI `AccountManager` care este programat să poată gestiona o varietate de diferite tipuri de cereri de modificare legate de conturile utilizatorilor.

Directiva sa sună astfel:

```
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.
```

Starea inițială a contului produsă de `account.to_directive` este pur și simplu o descriere textuală a contului, incluzând date relevante conexe precum utilizatori, abonamente etc.

Gama de funcții disponibile pentru `AccountManager` îi oferă capacitatea de a edita abonamentul utilizatorului, de a adăuga și elimina consultanți AI și alte

tipuri de componente adiționale plătite, precum și de a trimite e-mailuri de notificare proprietarului contului. Pe lângă funcția `finished`, acesta poate și să `notify_human_administrator` dacă întâmpină o eroare în timpul procesării sau necesită orice alt tip de asistență cu o solicitare.

Observați că în cazul unor întrebări, `AccountManager` poate alege să caute în baza de cunoștințe Olympia, unde poate găsi instrucțiuni despre cum să gestioneze cazurile limită și orice altă situație în care nu este sigur cum să procedeze.

Aplicaţii de Comerţ Electronic

În domeniul comerțului electronic, lucrătorii AI pot juca un rol crucial în îmbunătățirea experienței utilizatorului și optimizarea operațiunilor de afaceri. Iată câteva moduri în care lucrătorii AI pot fi utilizați:

Recomandări de Produse

Una dintre cele mai puternice aplicaţii ale lucrătorilor AI în comerţul electronic este generarea de recomandări personalizate de produse. Prin analizarea comportamentului utilizatorului, a istoricului achizițiilor și a preferințelor, aceşti lucrători pot sugera produse care sunt adaptate intereselor și nevoilor fiecărui utilizator în parte.

Cheia pentru recomandări eficiente de produse este utilizarea unei combinații de tehnici de filtrare colaborativă și filtrare bazată pe conținut. Filtrarea colaborativă analizează comportamentul utilizatorilor similari pentru a identifica tipare și a face recomandări bazate pe ce au cumpărat sau apreciat alții cu gusturi similare. Pe de altă parte, filtrarea bazată pe conținut se concentrează pe caracteristicile și atributele produselor în sine, recomandând articole care împărtășesc caracteristici similare cu cele față de care un utilizator a arătat interes anterior.

Iată un exemplu simplificat despre cum puteți implementa un lucrător de recomandări

de produse în Ruby, de această dată folosind un stil de programare funcțională "Railway Oriented (ROP)":

```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
```

Stilul de programare funcțională Ruby folosit în exemplu este influențat de F# și Rust. Puteți citi mai multe despre aceasta în explicația tehnicii oferită de prietenul meu Chad Wooley la GitLab

În acest exemplu, `ProductRecommendationWorker` primește un utilizator ca parametru de intrare și generează recomandări personalizate de produse prin transmiterea unui obiect de valoare printr-un lanț de pași funcționali. Să analizăm fiecare pas:

1. `ValidateUser.validate`: Acest pas asigură că utilizatorul este valid și eligibil pentru recomandări personalizate. Verifică dacă utilizatorul există, este activ și

are datele necesare disponibile pentru generarea recomandărilor. Dacă validarea eşuează, se returnează un rezultat de eroare şi lanţul este întrerupt.

2. `AnalyzeCurrentSession.analyze`: Dacă utilizatorul este valid, acest pas analizează sesiunea curentă de navigare a utilizatorului pentru a colecta informaţii contextuale. Acesta examinează interacţiunile recente ale utilizatorului, cum ar fi produsele vizualizate, căutările efectuate şi conţinutul coşului, pentru a înţelege interesele şi intenţiile curente.
3. `CollaborativeFilter.filter`: Folosind *comportamentul utilizatorilor similari*, acest pas aplică tehnici de filtrare colaborativă pentru a identifica produsele care ar putea fi de interes pentru utilizator. Ia în considerare factori precum istoricul achiziţiilor, evaluările şi interacţiunile utilizator-produs pentru a genera un set de recomandări candidate.
4. `ContentBasedFilter.filter`: Acest pas rafinează în continuare recomandările candidate prin aplicarea filtrării bazate pe conţinut. Compară atributele şi caracteristicile produselor candidate cu *preferinţele şi datele istorice ale utilizatorului* pentru a selecta elementele cele mai relevante.
5. `ProductSelector.select`: În final, acest pas selectează primele N produse din recomandările filtrate pe baza unor criterii predefinite, cum ar fi scorul de relevanţă, popularitatea sau alte reguli de business. Produsele selectate sunt apoi returnate ca recomandări personalizate finale.

Frumuseţea utilizării unui stil de programare funcţională în Ruby aici este că ne permite să înlănţuim aceşti paşi într-o manieră clară şi concisă. Fiecare pas se concentrează pe o sarcină specifică şi returnează un obiect `Result`, care poate fi fie un succes (`ok`), fie o eroare (`err`). Dacă oricare dintre paşi întâlneşte o eroare, lanţul este întrerupt, iar eroarea este propagată către rezultatul final.

În instrucţiunea `case` de la final, facem potrivirea tiparelor pe rezultatul final. Dacă rezultatul este o eroare (`ProductRecommendationError`), înregistrăm eroarea folosind un instrument precum Honeybadger pentru monitorizare şi depanare.

Dacă rezultatul este un succes (`ProductRecommendations`), transmitem un eveniment `:new_recommendations` folosind biblioteca de publicare/abonare Wisper, transmițând mai departe utilizatorul și recomandările generate.

Prin utilizarea tehnicilor de programare funcțională, putem crea un worker de recomandare a produselor modular și ușor de întreținut. Fiecare pas este independent și poate fi testat, modificat sau înlocuit cu ușurință fără a afecta fluxul general. Utilizarea potrivirii tiparelor și a clasei `Result` ne ajută să gestionăm erorile elegant și asigură că worker-ul eșuează rapid dacă orice pas întâmpină o problemă.

Desigur, acesta este un exemplu simplificat, iar într-un scenariu din lumea reală, ar trebui să vă integrați cu platforma de e-commerce, să gestionați cazurile limită și chiar să vă aventurați în implementarea algoritmilor de recomandare. Cu toate acestea, principiile de bază ale descompunerii problemei în pași mai mici și utilizarea tehnicilor de programare funcțională rămân aceleași.

Detectarea Fraudelor

Iată un exemplu simplificat despre cum puteți implementa un worker de detectare a fraudelor folosind același stil de Programare Orientată pe Șine (ROP) în Ruby:

```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
```

Clasa `FraudDetection` este un *obiect valoare* care încapsulează starea detectării fraudei pentru o anumită tranzacție. Aceasta oferă o modalitate structurată de a analiza și evalua riscul de fraudă asociat cu o tranzacție pe baza diferiților factori de risc.

```
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
```

```
27      risk_factors.values.sum
28    end
29  end
```

Clasa `FraudDetection` are următoarele atribute:

- `transaction`: O referință către tranzacția care este analizată pentru fraudă.
- `risk_factors`: Un array care stochează factorii de risc asociați cu tranzacția. Fiecare factor de risc este reprezentat ca un hash, unde cheia este descrierea factorului de risc, iar valoarea este probabilitatea de fraudă asociată cu acel factor de risc.

Metoda `add_risk_factor` permite adăugarea unui factor de risc în array-ul `risk_factors`. Aceasta primește doi parametri: `description`, care este un șir de caractere ce descrie factorul de risc, și `probability`, care este un număr cu virgulă mobilă ce reprezintă probabilitatea de fraudă asociată cu acel factor de risc. Folosim o condițională `case..in` pentru a efectua o verificare simplă a tipurilor.

Metoda `high_risk?` care va fi verificată la sfârșitul lanțului este o metodă predicat care compară `fraud_probability` (calculată prin însumarea probabilităților tuturor factorilor de risc) cu `RISK_THRESHOLD`.

Clasa `FraudDetection` oferă o modalitate curată și încapsulată de a gestiona detectarea fraudelor pentru o tranzacție. Aceasta permite adăugarea mai multor factori de risc, fiecare cu propria descriere și probabilitate, și oferă o metodă pentru a determina dacă tranzacția este considerată cu risc ridicat pe baza probabilității de fraudă calculate. Clasa poate fi ușor integrată într-un sistem mai mare de detectare a fraudelor, unde diferite componente pot colabora pentru a evalua și reduce riscul tranzacțiilor frauduloase.

În final, având în vedere că aceasta este o carte despre programare folosind AI, iată un exemplu de implementare a clasei `CheckCustomerHistory` care folosește procesarea AI utilizând modulul `ChatCompletion` din biblioteca mea Raix:

```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
```

În acest exemplu, `CheckCustomerHistory` definește o constantă `INSTRUCTION` care oferă instrucțiuni specifice modelului de IA despre cum să analizeze istoricul tranzacțiilor clientului pentru potențiali indicatori de fraudă prin intermediul unei directive de sistem

Metoda `self.check` este o metodă de clasă care inițializează o nouă instanță de `CheckCustomerHistory` cu obiectul `fraud_detection` și apelează metoda `call` pentru a efectua analiza istoricului clientului.

În interiorul metodei `call`, istoricul tranzacțiilor clientului este preluat și formatat într-un transcript care este transmis modelului de IA. Modelul de IA analizează istoricul tranzacțiilor pe baza instrucțiunilor furnizate și returnează un rezumat al constatărilor sale.

Constatările sunt adăugate la obiectul `fraud_detection`, iar obiectul `fraud_detection` actualizat este returnat ca un `Result` de succes.

Prin utilizarea modulului `ChatCompletion`, clasa `CheckCustomerHistory` poate folosi puterea IA pentru a analiza istoricul tranzacțiilor clientului și pentru a identifica potențiali indicatori de fraudă. Acest lucru permite tehnici de detectare a fraudei mai sofisticate și adaptive, deoarece modelul de IA poate învăța și se poate adapta la noi tipare și anomalii în timp.

Actualizarea clasei `FraudDetectionWorker` și a clasei `CheckCustomerHistory` demonstrează cum workerii bazați pe IA pot fi integrați fără probleme, îmbunătățind procesul de detectare a fraudei cu capacități de analiză și luare a deciziilor inteligente.

Analiza Sentimentului Clientului

Iată încă un exemplu similar despre cum puteți implementa un worker pentru analiza sentimentului clientului. De data aceasta cu mult mai puține explicații, deoarece ar trebui să înțelegeți deja cum funcționează acest stil de programare:

```
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
```

În acest exemplu, pașii `CustomerSentimentAnalysisWorker` includ preprocesarea feedback-ului (de exemplu, eliminarea zgomotului, tokenizarea), efectuarea analizei sentimentelor pentru a determina sentimentul general (pozitiv, negativ sau neutru), extragerea frazelor și subiectelor cheie, identificarea tendințelor și tiparelor și generarea de informații acționabile bazate pe analiză.

Aplicaţii în Domeniul Sănătăţii

În domeniul sănătății, lucrătorii AI pot asista profesioniștii medicali și cercetătorii în diverse sarcini, conducând la îmbunătățirea rezultatelor pacienților și accelerarea descoperirilor medicale. Iată câteva exemple:

Admiterea Pacienţilor

Lucrătorii AI pot eficientiza procesul de admitere a pacienților prin automatizarea diverselor sarcini și oferirea de asistență inteligentă.

Programarea Consultațiilor: Lucrătorii AI pot gestiona programarea consultațiilor prin înțelegerea preferințelor pacienților, a disponibilității și a urgenței nevoilor lor medicale. Aceștia pot interacționa cu pacienții prin interfețe conversaționale, ghidându-i prin procesul de programare și găsind cele mai potrivite intervale de timp bazate pe cerințele pacientului și disponibilitatea furnizorului de servicii medicale.

Colectarea Istoricului Medical: În timpul admiterii pacienților, lucrătorii AI pot asista în colectarea și documentarea istoricului medical al pacientului. Aceștia pot angaja dialoguri interactive cu pacienții, punând întrebări relevante despre afecțiunile medicale anterioare, medicamentele, alergiile și istoricul familial. Lucrătorii AI pot utiliza tehnici de procesare a limbajului natural pentru a interpreta și structura informațiile colectate, asigurându-se că acestea sunt captate cu acuratețe în fișa medicală electronică a pacientului.

Evaluarea și Stratificarea Simptomelor: Lucrătorii AI pot efectua evaluări inițiale ale simptomelor prin întrebarea pacienților despre simptomele actuale, durata, severitatea și orice factori asociați. Prin utilizarea bazelor de cunoștințe medicale și a modelelor de învățare automată, acești lucrători pot analiza informațiile furnizate și genera diagnostice diferențiale preliminare sau pot recomanda pașii următori adecvați, cum ar fi programarea unei consultații cu un furnizor de servicii medicale sau sugerarea măsurilor de auto-îngrijire.

Verificarea Asigurării: Lucrătorii AI pot asista în verificarea asigurării în timpul admiterii pacienților. Aceștia pot colecta detaliile asigurării pacientului, pot comunica cu furnizorii de asigurări prin API-uri sau servicii web și pot verifica eligibilitatea acoperirii și beneficiile. Această automatizare ajută la eficientizarea procesului de verificare a asigurării, reducând sarcina administrativă și asigurând captarea precisă a informațiilor.

Educația și Instrucțiunile pentru Pacienți: Lucrătorii AI pot furniza pacienților materiale educaționale relevante și instrucțiuni bazate pe afecțiunile lor medicale specifice sau procedurile viitoare. Aceștia pot livra conținut personalizat, pot răspunde la întrebări comune și pot oferi îndrumare privind pregătirile pre-consultație, instrucțiunile despre medicamente sau îngrijirea post-tratament. Acest lucru ajută la menținerea pacienților informați și implicați pe parcursul călătoriei lor medicale.

Prin utilizarea lucrătorilor AI în admiterea pacienților, organizațiile medicale pot îmbunătăți eficiența, reduce timpii de așteptare și îmbunătăți experiența generală a pacientului. Acești lucrători pot gestiona sarcini de rutină, pot colecta informații precise și pot oferi asistență personalizată, permițând profesioniștilor din domeniul medical să se concentreze pe furnizarea de îngrijiri de înaltă calitate pacienților.

Evaluarea Riscului Pacienților

Lucrătorii AI pot juca un rol crucial în evaluarea riscului pacienților prin analizarea diverselor surse de date și aplicarea tehnicilor avansate de analiză.

Integrarea Datelor: Lucrătorii AI pot colecta și interpreta date despre pacienți din multiple surse, cum ar fi fișele medicale electronice (FME), imagistica medicală, rezultatele de laborator, dispozitivele purtabile și determinanții sociali ai sănătății. Prin consolidarea acestor informații într-un profil cuprinzător al pacientului, lucrătorii AI pot oferi o vedere holistică asupra stării de sănătate și factorilor de risc ai pacientului.

Stratificarea Riscului: Lucrătorii AI pot utiliza modele predictive pentru a stratifica pacienții în diferite categorii de risc bazate pe caracteristicile lor individuale și datele

de sănătate. Această stratificare a riscului permite furnizorilor de servicii medicale să prioritizeze pacienții care necesită atenție sau intervenție mai imediată. De exemplu, pacienții identificați ca fiind cu risc ridicat pentru o anumită afecțiune pot fi marcați pentru monitorizare mai atentă, măsuri preventive sau intervenție timpurie.

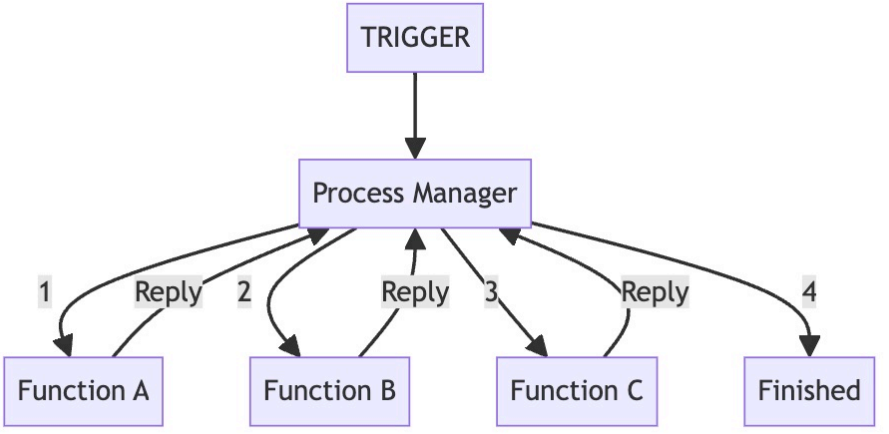
Profiluri de Risc Personalizate: Lucrătorii AI pot genera profiluri de risc personalizate pentru fiecare pacient, evidențiind factorii specifici care contribuie la scorurile lor de risc. Aceste profiluri pot include informații despre stilul de viață al pacientului, predispozițiile genetice, factorii de mediu și determinanții sociali ai sănătății. Prin furnizarea unei defalcări detaliate a factorilor de risc, lucrătorii AI pot ajuta furnizorii de servicii medicale să adapteze strategiile de prevenire și planurile de tratament la nevoile individuale ale pacienților.

Monitorizarea Continuă a Riscului: Lucrătorii AI pot monitoriza continuu datele pacienților și actualiza evaluările de risc în timp real. Pe măsură ce noi informații devin disponibile, cum ar fi modificări ale semnelor vitale, rezultatelor de laborator sau aderenței la medicație, lucrătorii AI pot recalcula scorurile de risc și alerta furnizorii de servicii medicale cu privire la orice modificări semnificative. Această monitorizare proactivă permite intervenții la timp și ajustări ale planurilor de îngrijire ale pacienților.

Suport pentru Decizii Clinice: Lucrătorii AI pot integra rezultatele evaluării riscului în sistemele de suport pentru decizii clinice, oferind furnizorilor de servicii medicale recomandări și alerte bazate pe dovezi. De exemplu, dacă scorul de risc al unui pacient pentru o anumită afecțiune depășește un anumit prag, lucrătorul AI poate sugera furnizorului de servicii medicale să ia în considerare teste diagnostice specifice, măsuri preventive sau opțiuni de tratament bazate pe ghiduri clinice și cele mai bune practici.

Acești lucrători pot procesa volume mari de date ale pacienților, pot aplica analize sofisticate și pot genera informații acționabile pentru a sprijini luarea deciziilor clinice. Acest lucru conduce în cele din urmă la îmbunătățirea rezultatelor pacienților, reducerea costurilor medicale și îmbunătățirea gestionării sănătății populației.

Lucrătorul AI ca Manager de Procese



În contextul aplicațiilor bazate pe AI, un lucrător poate fi proiectat să funcționeze ca un Manager de Procese, așa cum este descris în cartea "Enterprise Integration Patterns" de Gregor Hohpe. Un Manager de Procese este o componentă centrală care menține starea unui proces și determină următorii pași de procesare pe baza rezultatelor intermediare.

Când un lucrător AI acționează ca Manager de Procese, acesta primește un mesaj de intrare care inițializează procesul, cunoscut sub numele de *mesaj declanșator*. Lucrătorul AI menține apoi starea execuției procesului (ca transcript de conversație) și gestionează mesajul printr-o serie de pași de procesare implementați ca funcții utilitare, care pot fi secvențiale sau paralele, și apelate la discreția sa.

Dacă folosiți o clasă de model AI precum GPT-4 care știe cum să execute funcții în paralel, atunci lucrătorul dvs. poate executa mai mulți pași simultan. Ce-i drept, nu am încercat personal acest lucru și instinctul îmi spune că rezultatele pot varia.

După fiecare pas individual de procesare, controlul este returnat înapoi la lucrătorul AI, permițându-i să determine următorul/următorii pas(i) de procesare pe baza stării curente și a rezultatelor obținute.

Stocați Mesajele Declanșatoare

Din experiența mea, este înțelept să implementați mesajul declanșator ca un obiect susținut de bază de date. În acest fel, fiecare instanță a procesului este identificată printr-o cheie primară unică și vă oferă un loc pentru a stoca starea asociată cu execuția, inclusiv transcriptul conversației AI.

De exemplu, aici este o versiune simplificată a clasei model `AccountChange` a Olympiei, care reprezintă o cerere de a face o modificare în contul unui utilizator.

```
# == 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
```

Clasa `AccountChange` servește drept mesaj declanșator care inițiază un proces pentru gestionarea cererii de modificare a contului. Observați cum aceasta este transmisă către subsistemul de publicare-abonare bazat pe Wisper al Olympiei după ce tranzacția de creare se finalizează.

Stocarea mesajului declanșator în baza de date în acest mod oferă o înregistrare persistentă a fiecărei cereri de modificare a contului. Fiecare instanță a clasei `AccountChange` primește o cheie primară unică, permițând identificarea și urmărirea cu ușurință a cererilor individuale. Acest lucru este deosebit de util pentru scopuri de jurnalizare pentru audit, deoarece permite sistemului să mențină o înregistrare istorică a tuturor modificărilor conturilor, inclusiv momentul când au fost solicitate, ce modificări au fost cerute și starea curentă a fiecărei cereri.

În exemplul dat, clasa `AccountChange` include câmpuri precum `description` pentru a capta detaliile modificării solicitate, `state` pentru a reprezenta starea curentă a cererii (de exemplu, solicitată, completă, necesită_revizuire_umană) și `transcript` pentru a stoca transcriptul conversației cu AI-ul legat de cerere. Câmpul `description` este promptul efectiv care este utilizat pentru a iniția prima completare a conversației cu AI-ul. Stocarea acestor date oferă un context valoros și permite o mai bună urmărire și analiză a procesului de modificare a contului.

Stocarea mesajelor declanşatoare în baza de date permite o gestionare robustă a erorilor şi recuperare. Dacă apare o eroare în timpul procesării unei cereri de modificare a contului, sistemul marchează cererea ca eşuată şi o trece într-o stare care necesită intervenţie umană. Acest lucru asigură că nicio cerere nu este pierdută sau uitată, şi orice probleme pot fi abordate şi rezolvate corespunzător.

Lucrătorul AI, în calitate de Manager de Procese, oferă un punct central de control şi permite capabilităţi puternice de raportare şi depanare a proceselor. Cu toate acestea, este important de menţionat că utilizarea unui lucrător AI ca Manager de Procese pentru fiecare scenariu de flux de lucru din aplicaţia dumneavoastră poate fi exagerată.

Integrarea Lucrătorilor AI în Arhitectura Aplicaţiei Tale

Când incorporezi lucrători AI în arhitectura aplicaţiei tale, trebuie abordate mai multe considerente tehnice pentru a asigura o integrare fără probleme şi o comunicare eficientă între lucrătorii AI şi alte componente ale aplicaţiei. Această secţiune analizează aspectele cheie ale proiectării acestor interfeţe, gestionării fluxului de date şi administrării ciclului de viaţă al lucrătorilor AI.

Proiectarea Interfeţelor Clare şi a Protocoalelor de Comunicare

Pentru a facilita o integrare fără probleme între lucrătorii AI şi alte componente ale aplicaţiei, este crucial să definim interfeţe clare şi protocoale de comunicare. Ia în considerare următoarele abordări:

Integrare bazată pe API: Expune funcţionalitatea lucrătorilor AI prin API-uri bine definite, cum ar fi endpoint-uri RESTful sau scheme GraphQL. Acest lucru permite

altor componente să interacționeze cu lucrătorii AI folosind cereri și răspunsuri HTTP standard. Integrarea bazată pe API oferă un contract clar între lucrătorii AI și componentele care îi consumă, făcând mai ușoară dezvoltarea, testarea și mentenanța punctelor de integrare.

Comunicare bazată pe mesaje: Implementează modele de comunicare bazate pe mesaje, cum ar fi cozile de mesaje sau sistemele de publicare-abonare, pentru a permite interacțiunea asincronă între lucrătorii AI și alte componente. Această abordare decuplează lucrătorii AI de restul aplicației, permițând o mai bună scalabilitate, toleranță la erori și cuplare slabă. Comunicarea bazată pe mesaje este deosebit de utilă când procesarea efectuată de lucrătorii AI necesită mult timp sau resurse intensive, deoarece permite altor părți ale aplicației să continue execuția fără a aștepta ca lucrătorii AI să-și completeze sarcinile.

Arhitectură bazată pe evenimente: Proiectează-ți sistemul în jurul evenimentelor și declanșatoarelor care activează lucrătorii AI când sunt îndeplinite anumite condiții. Lucrătorii AI se pot abona la evenimente relevante și pot reacționa în consecință, efectuând sarcinile lor desemnate când apar evenimentele. Arhitectura bazată pe evenimente permite procesarea în timp real și permite ca lucrătorii AI să fie invocați la cerere, reducând consumul inutil de resurse. Această abordare este potrivită pentru scenarii în care lucrătorii AI trebuie să răspundă la acțiuni specifice sau modificări în starea aplicației.

Gestionarea Fluxului de Date și Sincronizarea

Când integrezi lucrători AI în aplicația ta, este crucial să asiguri un flux de date lin și să menții consistența datelor între lucrătorii AI și alte componente. Ia în considerare următoarele aspecte:

Pregătirea datelor: Înainte de a furniza date lucrătorilor AI, este posibil să fie necesară efectuarea diverselor sarcini de pregătire a datelor, cum ar fi curățarea, formatarea și/sau transformarea datelor de intrare. Nu doar vrei să te asiguri că lucrătorii AI pot procesa

eficient, dar vrei să te asiguri și că nu irosești token-uri acordând atenție informațiilor pe care lucrătorul le-ar putea considera inutile în cel mai bun caz, sau distractive în cel mai rău caz. Pregătirea datelor poate implica sarcini precum eliminarea zgomotului, gestionarea valorilor lipsă sau conversia tipurilor de date.

Persistența datelor: Cum vei stoca și persista datele care circulă înspre și dinspre lucrătorii AI? Ia în considerare factori precum volumul de date, modelele de interogare și scalabilitatea. Ai nevoie să persisti transcriptul AI-ului ca o reflecție a "procesului său de gândire" pentru scopuri de audit sau depanare, sau este suficient să ai doar o înregistrare a rezultatelor?

Recuperarea Datelor: Obținerea datelor necesare pentru workeri poate implica interogarea bazelor de date, citirea din fișiere sau accesarea API-urilor externe. Asigurați-vă că luați în considerare latența și modul în care workerii AI vor avea acces la cele mai actualizate date. Au nevoie de acces complet la baza de date sau ar trebui să definiți strict domeniul accesului lor în funcție de ceea ce fac? Dar scalabilitatea? Luați în considerare mecanismele de caching pentru a îmbunătăți performanța și a reduce încărcarea surselor de date subiacente.

Sincronizarea Datelor.: Când multiple componente, inclusiv workerii AI, accesează și modifică date partajate, este important să implementați mecanisme adecvate de sincronizare pentru a menține consistența datelor. Strategiile de blocare a bazelor de date, cum ar fi blocarea optimistă sau pesimistă, vă pot ajuta să preveniți conflictele și să asigurați integritatea datelor. Implementați tehnici de gestionare a tranzacțiilor pentru a grupa operațiunile de date conexe și a menține proprietățile ACID (atomicitate, consistență, izolare și durabilitate)

Gestionarea Erorilor și Recuperarea: Implementați mecanisme robuste de gestionare a erorilor și recuperare pentru a face față problemelor legate de date care pot apărea în timpul procesului de flux de date. Gestionați excepțiile cu eleganță și furnizați mesaje de eroare semnificative pentru a ajuta la depanare. Implementați mecanisme de reîncercare și strategii de rezervă pentru a gestiona eșecurile temporare sau întreruperile de rețea.

Definiți proceduri clare pentru recuperarea și restaurarea datelor în caz de corupere sau pierdere a datelor.

Prin proiectarea și implementarea atentă a mecanismelor de flux și sincronizare a datelor, puteți asigura că workerii AI au acces la date precise, consistente și actualizate. Acest lucru le permite să își îndeplinească sarcinile în mod eficient și să producă rezultate fiabile.

Gestionarea Ciclului de Viaţă al Workerilor AI

Dezvoltați un proces standardizat pentru inițializarea și configurarea workerilor AI. Prefer cadrele care standardizează modul în care definiți setările precum numele modelelor, directivele de sistem și definițiile funcțiilor. Asigurați-vă că procesul de inițializare este automatizat și reproductibil pentru a facilita implementarea și scalarea.

Implementați mecanisme cuprinzătoare de monitorizare și înregistrare pentru a urmări starea și performanța workerilor AI. Colectați metrici precum utilizarea resurselor, timpul de procesare, ratele de eroare și throughput-ul. Utilizați sisteme centralizate de logging precum ELK stack (Elasticsearch, Logstash, Kibana) pentru a agrega și analiza logurile de la mai mulți workeri AI.

Construiți toleranță la erori și reziliență în arhitectura workerilor AI. Implementați mecanisme de gestionare a erorilor și recuperare pentru a gestiona elegant eșecurile sau excepțiile. Modelele Mari de Limbaj sunt încă tehnologie de ultimă generație; furnizorii tind să se confrunte adesea cu întreruperi neașteptate. Utilizați mecanisme de reîncercare și întrerupătoare de circuit pentru a preveni eșecurile în cascadă.

Componibilitatea şi Orchestrarea Workerilor AI

Unul dintre avantajele cheie ale arhitecturii workerilor AI este componibilitatea sa, care vă permite să combinați și să orchestrați mai mulți workeri AI pentru a rezolva probleme

complexe. Prin descompunerea unei sarcini mai mari în subsarcini mai mici și mai ușor de gestionat, fiecare gestionată de un worker AI specializat, puteți crea sisteme puternice și flexibile. În această secțiune, vom explora diferite abordări pentru compunerea și orchestrarea "unei multitudini" de workeri AI.

Înlănţuirea Workerilor AI pentru Fluxuri de Lucru Multi-Etapă

În multe scenarii, o sarcină complexă poate fi descompusă într-o serie de pași secvențiali, unde ieșirea unui worker AI devine intrarea pentru următorul. Această înlănțuire a workerilor AI creează un flux de lucru sau un pipeline multi-etapă. Fiecare worker AI din lanț se concentrează pe o subsarcină specifică, iar ieșirea finală este rezultatul eforturilor combinate ale tuturor workerilor.

Să luăm în considerare un exemplu în contextul unei aplicații Ruby on Rails pentru procesarea conținutului generat de utilizatori. Fluxul de lucru implică următorii pași, care, ce-i drept, sunt probabil prea simpli pentru a merita descompunerea în acest mod în cazuri reale, dar fac exemplul mai ușor de înțeles:

1. **Curățarea Textului**: Un worker AI responsabil pentru eliminarea tag-urilor HTML, convertirea textului în litere mici și gestionarea normalizării Unicode.

2. **Detectarea Limbii**: Un worker AI care identifică limba textului curățat.

3. **Analiza Sentimentelor**: Un worker AI care determină sentimentul (pozitiv, negativ sau neutru) al textului bazat pe limba detectată.

4. **Categorizarea Conținutului**: Un worker AI care clasifică textul în categorii predefinite folosind tehnici de procesare a limbajului natural.

Iată un exemplu foarte simplificat despre cum puteți înlănțui acești workeri AI folosind 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
```

În acest exemplu, clasa `ContentProcessor` se inițializează cu textul brut și înlănțuie lucrătorii AI împreună în metoda `process`. Fiecare lucrător AI își îndeplinește sarcina specifică și transmite rezultatul către următorul lucrător din lanț. Rezultatul final este un hash care conține textul curățat, limba detectată, sentimentul și categoria conținutului.

Procesarea Paralelă pentru Lucrători AI Independenți

În exemplul anterior, lucrătorii AI sunt înlănțuiți secvențial, unde fiecare lucrător procesează textul și transmite rezultatul către următorul lucrător. Cu toate acestea, dacă aveți mai mulți lucrători AI care pot opera independent pe același input, puteți optimiza fluxul de lucru prin procesarea lor în paralel.

În scenariul dat, odată ce curățarea textului este efectuată de `TextCleanupWorker`, `LanguageDetectionWorker`, `SentimentAnalysisWorker` și `CategorizationWorker` pot procesa toți textul curățat în mod independent. Prin rularea acestor lucrători în paralel, puteți reduce potențial timpul total de procesare și îmbunătăți eficiența fluxului de lucru.

Pentru a realiza procesarea paralelă în Ruby, puteți folosi tehnici de concurență precum firele de execuție sau programarea asincronă. Iată un exemplu despre cum puteți

modifica clasa `ContentProcessor` pentru a procesa ultimii trei lucrători în paralel folosind fire de execuție:

```
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
```

În această versiune optimizată, folosim biblioteca `concurrent-ruby` pentru a crea obiecte `Concurrent::Future` pentru fiecare dintre workerii AI independenți. Un `Future` reprezintă un calcul care va fi executat asincron într-un fir de execuție separat.

După etapa de curățare a textului, creăm trei obiecte `Future`: `language_future`, `sentiment_future` și `category_future`. Fiecare `Future` execută workerul

AI corespunzător (`LanguageDetectionWorker`, `SentimentAnalysisWorker` şi `CategorizationWorker`) într-un fir de execuţie separat, transmiţând `cleaned_text` ca input.

Prin apelarea metodei `value` pentru fiecare `Future`, aşteptăm finalizarea calculului şi recuperăm rezultatul. Metoda `value` blochează execuţia până când rezultatul este disponibil, asigurându-ne că toţi workerii paraleli şi-au încheiat procesarea înainte de a continua.

În final, construim hash-ul de ieşire cu textul curăţat şi rezultatele de la workerii paraleli, la fel ca în exemplul original.

Prin procesarea workerilor AI independenţi în paralel, puteţi reduce potenţial timpul total de procesare în comparaţie cu rularea lor secvenţială. Această optimizare este deosebit de benefică când aveţi de-a face cu sarcini care consumă mult timp sau când procesaţi volume mari de date.

Cu toate acestea, este important de menţionat că câştigurile reale de performanţă depind de diverşi factori, cum ar fi complexitatea fiecărui worker, resursele de sistem disponibile şi costurile de gestionare a firelor de execuţie. Este întotdeauna o practică bună să faceţi evaluări comparative şi să profileţi codul pentru a determina nivelul optim de paralelism pentru cazul dumneavoastră specific.

În plus, când implementaţi procesarea paralelă, trebuie să fiţi atenţi la resursele partajate sau dependenţele dintre workeri. Asiguraţi-vă că workerii pot opera independent fără conflicte sau condiţii de cursă. Dacă există dependenţe sau resurse partajate, este posibil să fie nevoie să implementaţi mecanisme de sincronizare adecvate pentru a menţine integritatea datelor şi a evita probleme precum blocările reciproce sau rezultatele inconsistente.

Global Interpreter Lock (GIL) din Ruby şi Procesarea Asincronă

Este important să înțelegem implicațiile Global Interpreter Lock (GIL) din Ruby când luăm în considerare procesarea asincronă bazată pe fire de execuție în Ruby.

GIL este un mecanism în interpretorul Ruby care asigură că doar un singur fir de execuție poate executa cod Ruby la un moment dat, chiar și pe procesoare multi-core. Acest lucru înseamnă că, deși pot fi create și gestionate multiple fire de execuție într-un proces Ruby, doar un singur fir poate executa activ cod Ruby în orice moment.

GIL este proiectat pentru a simplifica implementarea interpretorului Ruby și pentru a oferi siguranță la nivel de fire de execuție pentru structurile de date interne ale Ruby. Cu toate acestea, limitează și potențialul de execuție cu adevărat paralelă a codului Ruby.

Când folosiți fire de execuție în Ruby, cum ar fi cu biblioteca `concurrent-ruby` sau clasa încorporată `Thread`, firele sunt supuse constrângerilor GIL. GIL permite fiecărui fir să execute cod Ruby pentru o scurtă perioadă de timp înainte de a comuta la alt fir, creând iluzia execuției concurente.

Cu toate acestea, din cauza GIL, execuția efectivă a codului Ruby rămâne secvențială. În timp ce un fir execută cod Ruby, celelalte fire sunt în esență puse în pauză, așteptând rândul lor pentru a obține GIL și a executa.

Acest lucru înseamnă că procesarea asincronă bazată pe fire de execuție în Ruby este cea mai eficientă pentru sarcinile dependente de I/O, cum ar fi așteptarea răspunsurilor API externe (cum ar fi modelele de limbaj mari găzduite de terți) sau efectuarea operațiilor I/O cu fișiere. Când un fir întâlnește o operație I/O, poate elibera GIL, permițând altor fire să execute în timp ce așteaptă finalizarea I/O.

Pe de altă parte, pentru sarcinile dependente de CPU, cum ar fi calculele intensive sau procesarea de lungă durată a workerilor AI, GIL poate limita potențialele câștiguri

de performanță ale paralelismului bazat pe fire de execuție. Deoarece doar un singur fir poate executa cod Ruby la un moment dat, timpul total de execuție poate să nu fie redus semnificativ în comparație cu procesarea secvențială.

Pentru a obține o execuție cu adevărat paralelă pentru sarcinile dependente de CPU în Ruby, poate fi necesar să explorați abordări alternative, cum ar fi:

- Utilizarea paralelismului bazat pe procese cu multiple procese Ruby, fiecare rulând pe un nucleu CPU separat.
- Folosirea bibliotecilor externe sau framework-urilor care oferă extensii native sau interfețe către limbaje fără GIL, cum ar fi C sau Rust.,
- Utilizarea framework-urilor de calcul distribuit sau a cozilor de mesaje pentru a distribui sarcinile între multiple mașini sau procese.

Este crucial să luați în considerare natura sarcinilor dumneavoastră și limitările impuse de GIL când proiectați și implementați procesarea asincronă în Ruby. În timp ce procesarea asincronă bazată pe fire de execuție poate oferi beneficii pentru sarcinile dependente de I/O, s-ar putea să nu ofere îmbunătățiri semnificative de performanță pentru sarcinile dependente de CPU din cauza constrângerilor GIL.

Tehnici de Ansamblu pentru Îmbunătățirea Acurateței

Tehnicile de ansamblu implică combinarea ieșirilor mai multor workeri AI pentru a îmbunătăți acuratețea generală sau robustețea sistemului. În loc să se bazeze pe un singur worker AI, tehnicile de ansamblu folosesc inteligența colectivă a mai multor workeri pentru a lua decizii mai bine informate.

Ansamblurile sunt deosebit de importante dacă diferite părți ale fluxului tău de lucru funcționează mai bine cu modele AI diferite, lucru mai frecvent decât ai putea crede. Modelele puternice precum GPT-4 sunt extrem de costisitoare în comparație cu opțiunile open source mai puțin capabile și probabil nu sunt necesare pentru fiecare etapă a fluxului de lucru din aplicația ta.

O tehnică comună de ansamblu este votul majorității, unde mai mulți lucrători AI procesează independent aceeași intrare, iar rezultatul final este determinat prin consensul majorității. Această abordare poate ajuta la atenuarea impactului erorilor individuale ale lucrătorilor și la îmbunătățirea fiabilității generale a sistemului.

Să luăm în considerare un exemplu în care avem trei lucrători AI pentru analiza sentimentelor, fiecare folosind un model diferit sau fiind prevăzut cu contexte diferite. Putem combina rezultatele lor folosind votul majorității pentru a determina predicția finală a sentimentului.

```
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
```

În acest exemplu, clasa `SentimentAnalysisEnsemble` se inițializează cu textul și apelează trei lucrători AI diferiți pentru analiza sentimentelor. Metoda `analyze` colectează predicțiile de la fiecare lucrător și determină sentimentul majoritar folosind metodele `group_by` și `max_by`. Rezultatul final este sentimentul care primește cele mai multe voturi din ansamblul de lucrători

Ansamblurile sunt în mod clar un caz în care merită să experimentezi cu paralelismul.

Selecția și Invocarea Dinamică a Lucrătorilor AI

În unele cazuri, dacă nu în majoritatea lor, lucrătorul AI specific care urmează să fie invocat poate depinde de condițiile de execuție sau de datele introduse de utilizator. Selecția și invocarea dinamică a lucrătorilor AI permite flexibilitate și adaptabilitate în sistem.

S-ar putea să te simți tentat să încerci să incluzi multe funcționalități într-un singur lucrător AI, oferindu-i multe funcții și un prompt complicat și voluminos care explică cum să le apeleze. Rezistă tentației, ai încredere în mine. Unul dintre motivele pentru care abordarea pe care o discutăm în acest capitol se numește "Multitudinea de Lucrători" este pentru a ne reaminti că este de dorit să avem mulți lucrători specializați, fiecare făcându-și propria mică sarcină în serviciul scopului mai mare.

De exemplu, să luăm în considerare o aplicație de tip chatbot în care diferiți lucrători AI sunt responsabili pentru gestionarea diferitelor tipuri de interogări ale utilizatorilor. Pe baza datelor introduse de utilizator, aplicația selectează dinamic lucrătorul AI potrivit pentru procesarea interogării.

```ruby
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
```

În acest exemplu, `ChatbotController` primește o interogare de la utilizator prin acțiunea `process_query`. Mai întâi folosește un `QueryClassifierWorker` pentru a determina tipul interogării. Pe baza tipului de interogare clasificat, controllerul selectează în mod dinamic lucrătorul AI potrivit pentru a genera răspunsul. Această selecție dinamică permite chatbotului să gestioneze diferite tipuri de interogări și să le direcționeze către lucrătorii AI relevanți.

Având în vedere că munca `QueryClassifierWorker` este relativ simplă și nu necesită mult context sau definiții de funcții, probabil îl puteți implementa folosind un LLM mic și ultra-rapid precum `mistralai/mixtral-8x7b-instruct:nitro`. Are capacități care se apropie de nivelul GPT-4 pentru multe sarcini și, la momentul când scriu acest lucru, Groq îl poate servi cu o viteză uluitoare de 444 de token-uri pe secundă.

Combinarea NLP Tradiţional cu LLM-uri

În timp ce Modelele de Limbaj de Mari Dimensiuni (LLM) au revoluționat domeniul procesării limbajului natural (NLP), oferind versatilitate și performanță fără precedent într-o gamă largă de sarcini, acestea nu sunt întotdeauna cea mai eficientă sau rentabilă soluție pentru fiecare problemă. În multe cazuri, combinarea tehnicilor tradiționale NLP cu LLM-uri poate duce la abordări mai optimizate, mai direcționate și mai economice pentru rezolvarea provocărilor specifice NLP.

Gândiți-vă la LLM-uri ca la briceagurile elvețiene ale NLP—incredibil de versatile și puternice, dar nu neapărat cel mai bun instrument pentru fiecare sarcină. Uneori, un instrument dedicat precum un tirbușon sau un desfăcător de conserve poate fi mai eficace și mai eficient pentru o sarcină specifică. În mod similar, tehnicile tradiționale NLP, precum gruparea documentelor, identificarea subiectelor și clasificarea, pot oferi adesea soluții mai direcționate și mai rentabile pentru anumite aspecte ale pipeline-ului NLP.

Unul dintre avantajele cheie ale tehnicilor tradiționale NLP este eficiența lor computațională. Aceste metode, care se bazează adesea pe modele statistice mai simple sau abordări bazate pe reguli, pot procesa volume mari de date text mult mai rapid și cu mai puține resurse computaționale în comparație cu LLM-urile. Acest lucru le face deosebit de potrivite pentru sarcini care implică analizarea și organizarea unor corpusuri mari de documente, cum ar fi gruparea articolelor similare sau identificarea subiectelor cheie într-o colecție de texte.

Mai mult, tehnicile tradiționale NLP pot atinge adesea acurațețe și precizie ridicate pentru sarcini specifice, mai ales când sunt antrenate pe seturi de date specifice domeniului. De exemplu, un clasificator de documente bine ajustat care folosește algoritmi tradiționali de învățare automată precum Mașini cu Vectori Suport (SVM) sau Naive Bayes poate categorisi cu acurațețe documentele în categorii predefinite cu un cost computațional minim.

Cu toate acestea, LLM-urile excelează cu adevărat când vine vorba de sarcini care necesită o înțelegere mai profundă a limbajului, contextului și raționamentului. Capacitatea lor de a genera text coerent și contextual relevant, de a răspunde la întrebări și de a rezuma pasaje lungi este neegalată de metodele tradiționale NLP. LLM-urile pot gestiona eficient fenomene lingvistice complexe, precum ambiguitatea, coreferința și expresiile idiomatice, făcându-le indispensabile pentru sarcini care necesită generare sau înțelegere de limbaj natural.

Adevărata putere constă în combinarea tehnicilor tradiționale NLP cu LLM-uri pentru a crea abordări hibride care valorifică punctele forte ale ambelor. Prin utilizarea metodelor tradiționale NLP pentru sarcini precum preprocesarea documentelor, gruparea și extragerea subiectelor, puteți organiza și structura eficient datele text. Aceste informații structurate pot fi apoi transmise către LLM-uri pentru sarcini mai avansate, precum generarea de rezumate, răspunsul la întrebări sau crearea de rapoarte comprehensive.

De exemplu, să luăm în considerare un caz de utilizare în care doriți să generați un raport de tendințe pentru un domeniu specific bazat pe un corpus mare de documente individuale despre tendințe. În loc să vă bazați exclusiv pe LLM-uri, care pot fi costisitoare din punct de vedere computațional și consumatoare de timp pentru procesarea unor volume mari de text, puteți folosi o abordare hibridă:

1. Utilizați tehnici tradiționale NLP, precum modelarea subiectelor (de exemplu, Alocare Latentă Dirichlet) sau algoritmi de grupare (de exemplu, K-means), pentru a grupa documente similare despre tendințe și pentru a identifica teme și subiecte cheie în corpus.
2. Transmiteți documentele grupate și subiectele identificate către un LLM, folosind capacitățile sale superioare de înțelegere și generare a limbajului pentru a crea rezumate coerente și informative pentru fiecare grup sau subiect.
3. În final, utilizați LLM-ul pentru a genera un raport comprehensiv de tendințe prin combinarea rezumatelor individuale, evidențiind cele mai semnificative tendințe

și oferind perspective și recomandări bazate pe informațiile agregate.

Prin combinarea tehnicilor tradiționale NLP cu LLM-uri în acest mod, puteți procesa eficient cantități mari de date text, extrage perspective semnificative și genera rapoarte de înaltă calitate, optimizând în același timp resursele computaționale și costurile.

Pe măsură ce vă lansați în proiectele de NLP, este esențial să evaluați cu atenție cerințele și constrângerile specifice ale fiecărei sarcini și să analizați modul în care metodele tradiționale de NLP și LLMs pot fi folosite împreună pentru a obține cele mai bune rezultate. Prin combinarea eficienței și preciziei tehnicilor tradiționale cu versatilitatea și puterea LLMs, puteți crea soluții NLP foarte eficiente și economice care aduc valoare utilizatorilor și părților interesate.

Utilizarea Instrumentelor

În domeniul dezvoltării aplicațiilor bazate pe IA, conceptul de "utilizare a instrumentelor" sau "apelare a funcțiilor" a apărut ca o tehnică puternică care permite modelului LLM să se conecteze la instrumente externe, API-uri, funcții, baze de date și alte resurse. Această abordare permite un set mai bogat de comportamente decât simpla generare de text și interacțiuni mai dinamice între componentele AI și restul ecosistemului aplicației. După cum vom analiza în acest capitol, utilizarea instrumentelor vă oferă și opțiunea de a face modelul AI să genereze date în moduri structurate.

Ce este Utilizarea Instrumentelor?

Utilizarea instrumentelor, cunoscută și sub numele de apelare a funcțiilor, este o tehnică care permite dezvoltatorilor să specifice o listă de funcții cu care un LLM poate interacționa în timpul procesului de generare. Aceste instrumente pot varia de

la simple funcții utilitare până la API-uri complexe sau interogări de baze de date. Prin oferirea accesului LLM-ului la aceste instrumente, dezvoltatorii pot extinde capacitățile modelului și îi pot permite să efectueze sarcini care necesită cunoștințe sau acțiuni externe.

Figura 7. Exemplu de definire a unei funcții pentru un lucrător AI care analizează documente

```
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
```

Ideea fundamentală din spatele utilizării instrumentelor este de a oferi LLM-ului capacitatea de a selecta și executa dinamic instrumentele potrivite în funcție de datele introduse de utilizator sau de sarcina în curs. În loc să se bazeze exclusiv pe cunoștințele pre-antrenate ale modelului, utilizarea instrumentelor permite LLM-ului să folosească

resurse externe pentru a genera răspunsuri mai precise, relevante și aplicabile. Utilizarea instrumentelor face ca tehnici precum RAG (Generare Augmentată prin Recuperare) să fie mult mai ușor de implementat decât ar fi în alte condiții.

> Rețineți că, dacă nu se specifică altfel, această carte presupune că modelul dvs. de AI nu are acces la niciun instrument integrat pe partea de server. Orice instrument pe care doriți să-l puneți la dispoziția AI-ului trebuie să fie declarat explicit de dvs. în fiecare cerere API, cu prevederi pentru executarea acestuia dacă și când AI-ul vă comunică că ar dori să utilizeze acel instrument în răspunsul său.

Potențialul Utilizării Instrumentelor

Utilizarea instrumentelor deschide o gamă largă de posibilități pentru aplicațiile bazate pe AI. Iată câteva exemple despre ce se poate realiza prin utilizarea instrumentelor:

1. **Chatboți și Asistenți Virtuali**: Prin conectarea unui LLM la instrumente externe, chatboții și asistenții virtuali pot efectua sarcini mai complexe, cum ar fi recuperarea informațiilor din baze de date, executarea apelurilor API sau interacțiunea cu alte sisteme. De exemplu, un chatbot ar putea utiliza un instrument CRM pentru a modifica starea unei oportunități în funcție de cererea utilizatorului.
2. **Analiza Datelor și Informații**: LLM-urile pot fi conectate la instrumente sau biblioteci de analiză a datelor pentru a efectua sarcini avansate de procesare a datelor. Acest lucru permite aplicațiilor să genereze informații, să efectueze analize comparative sau să ofere recomandări bazate pe date ca răspuns la interogările utilizatorilor.
3. **Căutare și Recuperare de Informații**: Utilizarea instrumentelor permite LLM-urilor să interacționeze cu motoare de căutare, baze de date vectoriale

sau alte sisteme de recuperare a informațiilor. Prin transformarea interogărilor utilizatorilor în interogări de căutare, LLM-ul poate recupera informații relevante din multiple surse și poate oferi răspunsuri complete la întrebările utilizatorilor.

4. **Integrarea cu Servicii Externe**: Utilizarea instrumentelor facilitează integrarea perfectă între aplicațiile bazate pe AI și serviciile sau API-urile externe. De exemplu, un LLM ar putea interacționa cu un API meteorologic pentru a furniza actualizări în timp real despre vreme sau cu un API de traducere pentru a genera răspunsuri multilingve.

Fluxul de Lucru pentru Utilizarea Instrumentelor

Fluxul de lucru pentru utilizarea instrumentelor implică de obicei patru pași cheie:

1. Includerea definițiilor funcțiilor în contextul cererii
2. Selectarea dinamică (sau explicită) a instrumentelor
3. Executarea funcției/funcțiilor
4. Continuarea opțională a promptului original

Să analizăm în detaliu fiecare dintre aceşti paşi.

Includerea definiţiilor funcţiilor în contextul cererii

AI-ul știe ce instrumente are la dispoziție deoarece îi oferiți o listă ca parte a cererii de completare (de obicei definită ca funcții folosind o variantă a schemei JSON).

Sintaxa exactă a definirii instrumentelor este specifică modelului.

Iată cum definiți o funcție `get_weather` în 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"]
    }
}
```

Și iată cum ai defini aceeași funcție pentru GPT-4, transmițând-o ca valoare a parametrului `tools`:

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

```
    },
}
```

> Aproape la fel, doar că diferit fără niciun motiv aparent! Ce enervant.

Definițiile funcțiilor specifică numele, descrierea și parametrii de intrare. Parametrii de intrare pot fi definiți mai detaliat folosind atribute precum enumerările pentru a limita valorile acceptabile și pentru a specifica dacă un parametru este obligatoriu sau nu.

Pe lângă definițiile efective ale funcțiilor, poți include, de asemenea, instrucțiuni sau context despre de ce și cum să folosești funcția în directiva de sistem.

De exemplu, instrumentul meu de Căutare Web din Olympia include această directivă de sistem, care îi reamintește IA-ului că are la dispoziție instrumentele menționate:

```
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.
```

Furnizarea unor descrieri detaliate este considerată cel mai important factor în performanța instrumentelor. Descrierile tale ar trebui să explice fiecare detaliu despre instrument, inclusiv:

- Ce face instrumentul
- Când ar trebui folosit (și când nu)
- Ce înseamnă fiecare parametru și cum afectează comportamentul instrumentului
- Orice precauții sau limitări importante care se aplică implementării instrumentului

Cu cât poți oferi mai mult context AI-ului despre instrumentele tale, cu atât acesta va fi mai bun în a decide când și cum să le folosească. De exemplu, Anthropic recomandă cel puțin 3-4 propoziții per descriere de instrument pentru seria sa Claude 3, mai mult dacă instrumentul este complex.

Nu este neapărat intuitiv, dar descrierile sunt considerate mai importante decât exemplele. În timp ce poți include exemple despre cum să folosești un instrument în descrierea sa sau în prompt-ul însoțitor, acest lucru este mai puțin important decât a avea o explicație clară și cuprinzătoare a scopului și parametrilor instrumentului. Adaugă exemple doar după ce ai dezvoltat complet descrierea.

Iată un exemplu de specificație de funcție API în stil Stripe:

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

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

În practică, unele modele au dificultăți în gestionarea specificațiilor funcțiilor imbricate și în tratarea tipurilor complexe de date de ieșire, cum ar fi array-urile, dicționarele etc. Dar teoretic, ar trebui să puteți furniza specificații JSON Schema de orice adâncime!

Selecţia Dinamică a Instrumentelor

Când executaţi o completare de chat care include definiţii de instrumente, modelul de limbaj alege în mod dinamic instrumentul(ele) cel(e) mai potrivit(e) şi generează parametrii de intrare necesari pentru fiecare instrument.

În practică, capacitatea AI-ului de a apela *exact* funcția corectă și de a urma *exact* specificația pentru parametrii de intrare este inconstantă. Reducerea parametrului de temperatură la 0.0 ajută mult, dar din experiența mea veți întâlni în continuare erori ocazionale. Aceste eșecuri includ nume de funcții halucinate, parametri de intrare denumiți greșit sau pur și simplu lipsă. Parametrii sunt transmiși ca JSON, ceea ce înseamnă că uneori veți vedea erori cauzate de JSON truncat, citat greșit sau deteriorat în alt mod.

Modelele de Date cu Auto-Vindecare pot ajuta la repararea automată a apelurilor de funcții care eșuează din cauza erorilor de sintaxă.

Selecția Forțată (sau Explicită) a Instrumentelor

Unele modele vă oferă opțiunea de a forța apelarea unei anumite funcții, ca parametru în cerere. În caz contrar, decizia de a apela sau nu funcția este lăsată complet la discreția AI-ului.

Capacitatea de a forța apelarea unei funcții este crucială în anumite scenarii unde doriți să vă asigurați că un instrument sau o funcție specifică este executată, indiferent de procesul de selecție dinamică al AI-ului. Există mai multe motive pentru care această capacitate este importantă:

1. **Control Explicit**: Este posibil să utilizați AI-ul ca o *Componentă Discretă* sau într-un flux de lucru predefinit care necesită executarea unei anumite funcții la un anumit moment. Prin forțarea apelului, puteți garanta că funcția dorită este invocată în loc să trebuiască să rugați frumos AI-ul să o facă.
2. **Depanare și Testare**: În timpul dezvoltării și testării aplicațiilor bazate pe AI, capacitatea de a forța apeluri de funcții este neprețuită pentru scopuri de depanare. Prin declanșarea explicită a funcțiilor specifice, puteți izola și testa componente individuale ale aplicației. Acest lucru vă permite să verificați corectitudinea

implementărilor funcțiilor, să validați parametrii de intrare și să vă asigurați că rezultatele așteptate sunt returnate.

3. **Gestionarea Cazurilor Limită**: Pot exista cazuri limită sau scenarii excepționale în care procesul de selecție dinamică al AI-ului ar putea să nu aleagă să execute o funcție pe care ar trebui să o execute, și știți asta bazându-vă pe procese externe. În astfel de cazuri, capacitatea de a forța un apel de funcție vă permite să gestionați explicit aceste situații. Definiți reguli sau condiții în logica aplicației pentru a determina când să suprascrieți discreția AI-ului.
4. **Consistență și Reproductibilitate**: Dacă aveți o secvență specifică de funcții care trebuie executate într-o anumită ordine, forțarea apelurilor garantează că aceeași secvență este urmată de fiecare dată. Acest lucru este deosebit de important în aplicațiile unde consistența și comportamentul predictibil sunt critice, cum ar fi în sistemele financiare sau simulările științifice.
5. **Optimizarea Performanței**: În unele cazuri, forțarea unui apel de funcție poate duce la optimizări de performanță. Dacă știți că o anumită funcție este necesară pentru o sarcină specifică și că procesul de selecție dinamică al AI-ului ar putea introduce overhead inutil, puteți ocoli procesul de selecție și invoca direct funcția necesară. Acest lucru poate ajuta la reducerea latenței și îmbunătățirea eficienței generale a aplicației.

Pe scurt, capacitatea de a forța apeluri de funcții în aplicațiile bazate pe AI oferă control explicit, ajută la depanare și testare, gestionează cazuri limită, asigură consistență și reproductibilitate. Este un instrument puternic în arsenalul dumneavoastră, dar trebuie să discutăm încă un aspect al acestei caracteristici importante.

În multe cazuri de luare a deciziilor, vrem întotdeauna ca modelul să facă un apel de funcție și este posibil să nu vrem niciodată ca modelul să răspundă doar cu cunoștințele sale interne. De exemplu, dacă direcționați între multiple modele specializate în sarcini diferite (input multilingv, matematică, etc), puteți utiliza modelul de apelare a funcțiilor pentru a delega cereri către unul dintre modelele ajutătoare și niciodată să nu răspundă independent.

Parametrul de Alegere a Instrumentului

GPT-4 și alte modele de limbaj care suportă apelarea funcțiilor vă oferă un parametru `tool_choice` pentru controlul asupra utilizării obligatorii a instrumentelor ca parte a unei completări. Acest parametru are trei valori posibile:

- `auto` oferă AI-ului discreție completă asupra utilizării unui instrument sau simplu răspuns
- `required` spune AI-ului că *trebuie* să apeleze un instrument *în loc* să răspundă, dar lasă selecția instrumentului la latitudinea AI-ului
- A treia opțiune este să setați parametrul cu `name_of_function` pe care doriți să îl forțați. Mai multe despre asta în secțiunea următoare.

Rețineți că dacă setați tool choice la `required`, modelul va fi forțat să aleagă cea mai relevantă funcție dintre cele furnizate, chiar dacă niciuna nu se potrivește cu adevărat promptului. La momentul publicării, nu cunosc niciun model care să returneze un răspuns `tool_calls` gol sau să folosească alte modalități de a vă anunța că nu a găsit o funcție potrivită de apelat.

Forţarea unei Funcţii pentru a Obţine Date Structurate

Abilitatea de a forța un apel de funcție vă oferă o modalitate de a obține date structurate dintr-o completare de chat în loc să fie nevoie să le extrageți singur din răspunsul în text simplu.

> De ce este importantă forțarea funcțiilor pentru a obține date structurate? Simplu spus, pentru că extragerea datelor structurate din output-ul unui LLM este o bătaie de cap. Vă puteți face viața puțin mai ușoară cerând datele în XML, dar apoi trebuie să parseați XML-ul. Și ce faceți când acel XML lipsește pentru că IA v-a răspuns: "Îmi pare rău, dar nu pot genera datele solicitate deoarece bla, bla, bla..."

Când folosiți uneltele în acest mod:

- Ar trebui probabil să definiți o singură unealtă în cererea dvs.
- Nu uitați să forțați utilizarea funcției sale folosind parametrul `tool_choice`.
- Țineți minte că modelul va transmite input-ul către unealtă, așa că numele uneltei și descrierea ar trebui să fie din perspectiva modelului, nu a dvs.

Acest ultim punct merită un exemplu pentru clarificare. Să zicem că cereți IA-ului să facă analiza sentimentelor pe textul utilizatorului. Numele funcției nu ar fi `analyze_sentiment`, ci mai degrabă ceva de genul `save_sentiment_analysis`. IA-ul este cel care face analiza sentimentelor, *nu unealta.* Tot ce face unealta (din perspectiva IA-ului) este să salveze rezultatele analizei.

Iată un exemplu de utilizare a Claude 3 pentru a înregistra un rezumat al unei imagini în JSON bine structurat, de data aceasta din linia de comandă folosind `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."
                }
            ]
        }
    ]
}'
```

În exemplul furnizat, folosim modelul Claude 3 de la Anthropic pentru a genera un rezumat JSON structurat al unei imagini. Iată cum funcționează:

1. Definim un singur instrument numit `record_summary` în array-ul `tools` din payload-ul cererii. Acest instrument este responsabil pentru înregistrarea unui rezumat al imaginii într-un JSON bine structurat.
2. Instrumentul `record_summary` are o `input_schema` care specifică structura aşteptată a output-ului JSON. Aceasta defineşte trei proprietăţi:

    - `key_colors`: Un array de obiecte care reprezintă culorile cheie din imagine. Fiecare obiect de culoare are proprietăţi pentru valorile de roşu, verde şi albastru (variind de la 0.0 la 1.0) şi un nume de culoare lizibil în format snake_case.
    - `description`: O proprietate de tip string pentru o scurtă descriere a imaginii, limitată la 1-2 propoziţii.
    - `estimated_year`: O proprietate opţională de tip întreg pentru anul estimat în care a fost făcută fotografia, dacă pare să fie o fotografie non-ficţională.

3. În array-ul `messages`, furnizăm datele imaginii ca un şir codificat base64 împreună cu tipul media. Acest lucru permite modelului să proceseze imaginea ca parte a input-ului.
4. De asemenea, îi cerem lui Claude să folosească instrumentul `record_summary` pentru a descrie imaginea.
5. Când cererea este trimisă către modelul Claude 3, acesta analizează imaginea şi generează un rezumat JSON bazat pe `input_schema` specificată. Modelul extrage culorile cheie, oferă o scurtă descriere şi estimează anul în care a fost făcută fotografia (dacă este cazul).
6. Rezumatul JSON generat este transmis ca parametri către instrumentul `record_summary`, oferind o reprezentare structurată a caracteristicilor cheie ale imaginii.

Prin utilizarea instrumentului `record_summary` cu o `input_schema` bine definită, putem obţine un rezumat JSON structurat al unei imagini fără a ne baza pe extragerea de text simplu. Această abordare asigură că output-ul urmează un format consistent şi poate fi uşor analizat şi procesat de componentele din aval ale aplicaţiei.

Capacitatea de a forța un apel de funcție și de a specifica structura așteptată a output-ului este o caracteristică puternică a utilizării instrumentelor în aplicațiile bazate pe AI. Aceasta le permite dezvoltatorilor să aibă mai mult control asupra output-ului generat și simplifică integrarea datelor generate de AI în fluxul de lucru al aplicației lor.

Executarea Funcţiei(lor)

Ai definit funcții și ți-ai instruit AI-ul, care a decis că ar trebui să apeleze una dintre funcțiile tale. Acum este momentul ca codul tău de aplicație sau biblioteca, dacă folosești un gem Ruby precum `raix-rails` să direcționeze apelul funcției și parametrii acesteia către implementarea corespunzătoare *în codul tău de aplicație.*

Codul tău de aplicație decide ce să facă cu rezultatele execuției funcției. Poate că acest lucru implică o singură linie de cod într-o lambda, sau poate implică apelarea unui API extern. Poate implică apelarea unei alte componente AI, sau poate implică sute sau chiar mii de linii de cod în restul sistemului tău. Depinde în întregime de tine.

Uneori apelul funcției este sfârșitul operațiunii, dar dacă rezultatele reprezintă informații într-un lanț de gândire care trebuie continuat de AI, atunci codul tău de aplicație trebuie să insereze rezultatele execuției în transcriptul de chat și să permită AI-ului să continue procesarea.

De exemplu, aici avem o declarație de funcție Raix folosită de `AccountManager`-ul Olympia pentru a comunica cu clienții noștri ca parte a unei Orchestrări Inteligente a Fluxului de Lucru pentru serviciul clienți.

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

  # lots of other functions...

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

Este posibil să nu fie imediat clar ce se întâmplă aici, așa că voi descompune explicația.

1. Clasa `AccountManager` definește multe funcții legate de gestionarea contului. Poate schimba planul tău, poate adăuga și elimina membri ai echipei, printre alte lucruri.
2. Instrucțiunile sale de nivel superior îi spun lui `AccountManager` că ar trebui să notifice proprietarul contului cu rezultatele cererii de modificare a contului, folosind funcția `notify_account_owner`.
3. Definiția concisă a funcției include:

- numele
- descrierea
- parametrii `message: { type: "string" }`
- un bloc de cod care va fi executat când funcția este apelată

După actualizarea transcriptului cu rezultatele blocului funcției, metoda `chat_completion` este apelată din nou. Această metodă este responsabilă pentru trimiterea transcriptului actualizat al conversației înapoi către modelul AI pentru procesare ulterioară. Ne referim la acest proces ca la o *buclă de conversație*.

Când modelul AI primește o nouă cerere de completare a chat-ului cu un transcript actualizat, acesta are acces la rezultatele funcției executate anterior. Poate analiza aceste rezultate, le poate încorpora în procesul său decizional și poate genera următorul răspuns sau acțiune bazată pe contextul cumulativ al conversației. Poate alege să execute funcții suplimentare bazate pe contextul actualizat, sau poate genera un răspuns final la promptul original dacă determină că nu mai sunt necesare alte apeluri de funcții.

Continuarea Opțională a Promptului Original

Când trimiți rezultatele instrumentului înapoi către LLM și continui procesarea promptului original, AI-ul folosește acele rezultate fie pentru a apela funcții suplimentare, fie pentru a genera un răspuns final în text simplu.

Unele modele, precum Command-R de la Cohere, pot cita instrumentele specifice pe care le-au folosit în răspunsurile lor, oferind transparență și trasabilitate suplimentară.

În funcție de modelul utilizat, rezultatele apelului funcției vor exista în mesajele transcriptului care au propriul rol special sau vor fi reflectate într-o altă sintaxă. Dar partea importantă este ca acele date să fie în transcript, astfel încât să poată fi luate în considerare de AI atunci când decide ce să facă în continuare.

O eroare comună (și potențial costisitoare) este să uiți să adaugi rezultatele funcției în transcript înainte de a continua chat-ul. În consecință, AI-ul va primi prompt-ul în esență în același mod în care l-a primit înainte să apeleze funcția prima dată. Cu alte cuvinte, din punctul de vedere al AI-ului, încă nu a apelat funcția. Așa că o apelează din nou. Și din nou. Și din nou, la nesfârșit până când îl întrerupi. Sperăm că contextul tău nu a fost prea mare și modelul nu a fost prea scump!

Cele Mai Bune Practici pentru Utilizarea Instrumentelor

Pentru a obține maximum de la utilizarea instrumentelor, ia în considerare următoarele practici recomandate.

Definiţii Descriptive

Oferă nume și descrieri clare și descriptive pentru fiecare instrument și parametrii săi de intrare. Acest lucru ajută LLM-ul să înțeleagă mai bine scopul și capacitățile fiecărui instrument.

> Pot să vă spun din experiență că înțelepciunea comună care spune că "denumirea este dificilă" se aplică și aici; am văzut rezultate dramatic diferite de la LLM-uri doar prin schimbarea numelor funcțiilor sau formularea descrierilor. Uneori, eliminarea descrierilor *îmbunătățește* performanța.

Procesarea Rezultatelor Instrumentelor

Când transmiți rezultatele instrumentelor înapoi către LLM, asigură-te că acestea sunt bine structurate și cuprinzătoare. Folosește chei și valori semnificative pentru a reprezenta ieșirea fiecărui instrument. Experimentează cu diferite formate și vezi care funcționează cel mai bine, de la JSON la text simplu.

Interpretorul de Rezultate abordează această provocare prin utilizarea AI-ului pentru a analiza rezultatele și a oferi explicații prietenoase pentru utilizatori, rezumate sau concluzii cheie.

Gestionarea Erorilor

Implementează mecanisme robuste de gestionare a erorilor pentru a trata cazurile în care LLM-ul poate genera parametri de intrare invalizi sau neacceptați pentru apelurile instrumentelor. Gestionează și recuperează-te elegant din orice erori care pot apărea în timpul execuției instrumentului.

O calitate extrem de plăcută a AI-ului este că înțelege mesajele de eroare! Ceea ce înseamnă că dacă lucrezi într-o mentalitate rapidă și aproximativă, poți pur și simplu să prinzi orice excepții generate în implementarea unui instrument și să le transmiți înapoi către AI astfel încât să știe ce s-a întâmplat!

De exemplu, iată o versiune simplificată a implementării căutării Google în Olympia:

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

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

Căutările Google în Olympia reprezintă un proces în doi pași. Mai întâi efectuezi căutarea, apoi rezumi rezultatele. Dacă apare o eroare, indiferent care ar fi aceasta, mesajul de excepție este împachetat și trimis înapoi către AI. Această tehnică este fundamentul practic al tuturor modelelor de *Gestionare Inteligentă a Erorilor*.

De exemplu, să presupunem că apelul API `GoogleSearch` eșuează din cauza unei excepții 503 Service Unavailable. Aceasta se propagă până la nivelul superior de captare a erorilor, iar descrierea erorii este trimisă înapoi către AI ca rezultat al apelului funcției. În loc să îi arate utilizatorului un ecran gol sau o eroare tehnică, AI-ul spune ceva de

genul “Îmi pare rău, dar nu pot accesa capacitățile mele de Căutare Google în acest moment. Pot încerca din nou mai târziu, dacă doriți.”

Acest lucru poate părea doar un truc inteligent, dar gândiți-vă la un alt tip de eroare, unul în care AI-ul apela un API extern și avea control direct asupra parametrilor transmiși către API. Poate a făcut o greșeală în modul în care a generat acești parametri? Cu condiția ca mesajul de eroare de la API-ul extern să fie suficient de detaliat, transmiterea mesajului de eroare înapoi către AI-ul apelant înseamnă că acesta poate reconsidera acei parametri și poate încerca din nou. Automat. Indiferent care ar fi fost eroarea.

Acum gândiți-vă ce ar fi necesar pentru a replica acest tip de gestionare robustă a erorilor în cod *normal*. Este practic imposibil.

Rafinare Iterativă

Dacă LLM-ul nu recomandă instrumentele adecvate sau generează răspunsuri suboptime, iterați asupra definițiilor instrumentelor, descrierilor și parametrilor de intrare. Rafinați și îmbunătățiți continuu configurarea instrumentelor pe baza comportamentului observat și a rezultatelor dorite.

1. Începeți cu definiții simple ale instrumentelor: Începeți prin definirea instrumentelor cu nume, descrieri și parametri de intrare clare și concise. Evitați să complicați inițial configurarea instrumentelor și concentrați-vă pe funcționalitatea de bază. De exemplu, dacă doriți să salvați rezultatele analizei sentimentelor, începeți cu o definiție de bază precum:

```
{
  "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. Testează şi observă: După ce ai definiţiile iniţiale ale instrumentelor implementate, testează-le cu diferite prompturi şi observă cum interacţionează LLM-ul cu instrumentul. Acordă atenţie calităţii şi relevanţei răspunsurilor generate. Dacă LLM-ul generează răspunsuri suboptime, este momentul să rafinezi definiţiile instrumentelor.
3. Rafinează descrierile: Dacă LLM-ul înţelege greşit scopul unui instrument, încearcă să rafinezi descrierea instrumentului. Oferă mai mult context, exemple sau clarificări pentru a ghida LLM-ul în utilizarea eficientă a instrumentului. De exemplu, poţi actualiza descrierea instrumentului de analiză a sentimentelor pentru a aborda mai specific *tonul emoţional* al textului analizat:

```
{
  "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. Ajustaţi parametrii de intrare: Dacă LLM-ul generează parametri de intrare invalizi sau irelevanţi pentru un instrument, luaţi în considerare ajustarea

definițiilor parametrilor. Adăugați constrângeri mai specifice, reguli de validare sau exemple pentru a clarifica formatul de intrare așteptat.

5. Iterați pe baza feedback-ului: Monitorizați continuu performanța instrumentelor și colectați feedback de la utilizatori sau părți interesate. Folosiți acest feedback pentru a identifica zonele care necesită îmbunătățiri și faceți rafinări iterative la definițiile instrumentelor. De exemplu, dacă utilizatorii raportează că analiza nu gestionează bine sarcasmul, puteți adăuga o notă în descriere:

```
{
  "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.",
  ...
}
```

Prin rafinarea iterativă a definițiilor instrumentelor bazată pe comportamentul observat și feedback, puteți îmbunătăți treptat performanța și eficacitatea aplicației dumneavoastră bazate pe IA. Nu uitați să păstrați definițiile instrumentelor clare, concise și concentrate pe sarcina specifică în cauză. Testați și validați în mod regulat interacțiunile instrumentelor pentru a vă asigura că se aliniază cu rezultatele dorite.

Compunerea și Înlănțuirea Instrumentelor

Unul dintre cele mai puternice aspecte ale utilizării instrumentelor, care a fost doar menționat până acum, este capacitatea de a compune și înlănțui multiple instrumente pentru a realiza sarcini complexe. Prin proiectarea atentă a definițiilor instrumentelor și a formatelor lor de intrare/ieșire, puteți crea blocuri de construcție reutilizabile care pot fi combinate în diverse moduri.

Să luăm în considerare un exemplu în care construiți un flux de analiză a datelor pentru aplicația dumneavoastră bazată pe IA. Ați putea avea următoarele instrumente:

1. **`DataRetrieval`**: Un instrument care extrage date dintr-o bază de date sau API pe baza unor criterii specificate.
2. **`DataProcessing`**: Un instrument care efectuează calcule, transformări sau agregări asupra datelor extrase.
3. **`DataVisualization`**: Un instrument care prezintă datele procesate într-un format prietenos pentru utilizator, cum ar fi diagrame sau grafice.

Prin înlănțuirea acestor instrumente, puteți crea un flux de lucru puternic care extrage datele relevante, le procesează și prezintă rezultatele într-un mod semnificativ. Iată cum ar putea arăta fluxul de lucru al utilizării instrumentelor:

1. LLM-ul primește o interogare de la utilizator care solicită informații despre datele de vânzări pentru o anumită categorie de produse.
2. LLM-ul selectează instrumentul `DataRetrieval` și generează parametrii de intrare adecvați pentru a extrage datele relevante despre vânzări din baza de date.
3. Datele extrase sunt "transmise" către instrumentul `DataProcessing`, care calculează metrici precum venitul total, prețul mediu de vânzare și rata de creștere.
4. Datele procesate sunt apoi digerate de instrumentul `DataVisualization`, care creează o diagramă sau un grafic vizual atractiv pentru a reprezenta informațiile, transmițând URL-ul diagramei înapoi către LLM.
5. În final, LLM-ul generează un răspuns formatat la interogarea utilizatorului folosind markdown, încorporând datele vizualizate și oferind un rezumat al constatărilor principale.

Prin compunerea acestor instrumente, puteți crea un flux de lucru fluid de analiză a datelor care poate fi ușor integrat în aplicația dumneavoastră. Frumusețea acestei abordări este că fiecare instrument poate fi dezvoltat și testat independent, iar apoi combinat în diferite moduri pentru a rezolva diverse probleme.

Pentru a permite o compunere și înlănțuire fluidă a instrumentelor, este important să definiți formate clare de intrare și ieșire pentru fiecare instrument.

De exemplu, instrumentul `DataRetrieval` ar putea accepta parametri precum detaliile de conexiune la baza de date, numele tabelului și condițiile de interogare, și să returneze setul de rezultate ca un obiect JSON structurat. Instrumentul `DataProcessing` poate apoi să aștepte acest obiect JSON ca intrare și să producă un obiect JSON transformat ca ieșire. Prin standardizarea fluxului de date între instrumente, puteți asigura compatibilitatea și reutilizabilitatea.

În timp ce vă proiectați ecosistemul de instrumente, gândiți-vă la modul în care diferite instrumente pot fi combinate pentru a aborda cazurile de utilizare comune în aplicația dumneavoastră. Luați în considerare crearea unor instrumente de nivel înalt care încapsulează fluxuri de lucru comune sau logică de afaceri, făcând mai ușor pentru LLM să le selecteze și să le utilizeze eficient.

Amintiți-vă, puterea utilizării instrumentelor constă în flexibilitatea și modularitatea pe care le oferă. Prin descompunerea sarcinilor complexe în instrumente mai mici și reutilizabile, puteți crea o aplicație robustă și adaptabilă bazată pe IA care poate face față unei game largi de provocări.

Direcţii Viitoare

Pe măsură ce domeniul dezvoltării aplicațiilor bazate pe IA evoluează, putem aștepta progrese suplimentare în capacitățile de utilizare a instrumentelor. Câteva direcții viitoare potențiale includ:

1. **Utilizarea Multi-hop a Instrumentelor**: LLM-urile ar putea fi capabile să decidă de câte ori trebuie să folosească instrumentele pentru a genera un răspuns satisfăcător. Acest lucru ar putea implica mai multe runde de selecție și execuție a instrumentelor bazate pe rezultate intermediare.

2. **Instrumente Predefinite**: Platformele de IA ar putea oferi un set de instrumente predefinite pe care dezvoltatorii le pot folosi direct, cum ar fi interpreți Python, instrumente de căutare web sau funcții utilitare comune.
3. **Integrare Fluidă**: Pe măsură ce utilizarea instrumentelor devine mai răspândită, putem aștepta o mai bună integrare între platformele de IA și cadrele de dezvoltare populare, făcând mai ușor pentru dezvoltatori să încorporeze utilizarea instrumentelor în aplicațiile lor.

Utilizarea instrumentelor este o tehnică puternică care permite dezvoltatorilor să valorifice întregul potențial al LLM-urilor în aplicațiile bazate pe IA. Prin conectarea LLM-urilor la instrumente și resurse externe, puteți crea sisteme mai dinamice, inteligente și conștiente de context care se pot adapta la nevoile utilizatorilor și pot oferi informații și acțiuni valoroase.

În timp ce utilizarea instrumentelor oferă posibilități imense, este important să fim conștienți de potențialele provocări și considerente. Un aspect cheie este gestionarea complexității interacțiunilor dintre instrumente și asigurarea stabilității și fiabilității sistemului general. Trebuie să gestionați scenarii în care apelurile instrumentelor pot eșua, pot returna rezultate neașteptate sau pot avea implicații asupra performanței. În plus, ar trebui să luați în considerare măsuri de securitate și control al accesului pentru a preveni utilizarea neautorizată sau malițioasă a instrumentelor. Mecanismele adecvate de gestionare a erorilor, jurnalizare și monitorizare sunt cruciale pentru menținerea integrității și performanței aplicației dumneavoastră bazate pe IA.

Pe măsură ce explorați posibilitățile utilizării instrumentelor în propriile proiecte, amintiți-vă să începeți cu obiective clare, să proiectați definiții bine structurate ale instrumentelor și să iterați pe baza feedback-ului și a rezultatelor. Cu abordarea și mentalitatea potrivită, utilizarea instrumentelor poate debloca noi niveluri de inovație și valoare în aplicațiile bazate pe IA

Procesarea Fluxurilor

Transmiterea datelor în flux prin HTTP, cunoscută și sub numele de evenimente trimise de server (SSE), este un mecanism prin care serverul trimite continuu date către client pe măsură ce acestea devin disponibile, fără ca clientul să trebuiască să le solicite explicit. Având în vedere că răspunsul IA este generat incremental, are sens să oferim o experiență de utilizare receptivă prin afișarea rezultatului IA pe măsură ce acesta este generat. De fapt, toate API-urile furnizorilor de IA pe care le cunosc oferă răspunsuri în flux ca opțiune în punctele lor finale de completare.

Motivul pentru care acest capitol apare aici în carte, imediat după Utilizarea Instrumentelor, este datorită potențialului puternic pe care îl are combinarea utilizării instrumentelor cu răspunsurile IA în timp real către utilizatori. Acest lucru permite experiențe dinamice și interactive în care IA poate procesa datele de intrare ale utilizatorului, poate utiliza diverse instrumente și funcții la discreția sa și poate oferi răspunsuri în timp real.

Pentru a realiza această interacțiune fluidă, trebuie să scrieți gestionari de flux care pot distribui atât apelurile funcțiilor utilitare invocate de IA, cât și text simplu către utilizatorul final. Necesitatea de a relua procesarea după executarea unei funcții utilitare adaugă o provocare interesantă sarcinii.

Implementarea unui ReplyStream

Pentru a demonstra modul în care poate fi implementată procesarea fluxurilor, acest capitol va analiza în profunzime o versiune simplificată a clasei `ReplyStream` care este utilizată în Olympia. Instanțele acestei clase pot fi transmise ca parametrul `stream` în bibliotecile client IA precum ruby-openai și openrouter

Iată cum folosesc `ReplyStream` în `PromptSubscriber` al Olympia, care ascultă prin intermediul Wisper crearea de noi mesaje ale utilizatorului.

```
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...
```

Pe lângă o referință `context` către abonatul prompt care a instanțiat-o, clasa `ReplyStream` are și variabile de instanță pentru a stoca un buffer de date primite, precum și array-uri pentru a ține evidența numelor de funcții și argumentelor invocate în timpul procesării fluxului.

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

  delegate :bot_message, :dispatch, to: :context

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

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

  # ...
end
```

Metoda `initialize` configurează starea inițială a instanței `ReplyStream`, inițializând buffer-ul, contextul și alte variabile.

Metoda `call` este punctul principal de intrare pentru procesarea datelor în flux. Aceasta primește un `chunk` de date (reprezentat ca un hash) și un parametru opțional `bytesize`, care în exemplul nostru nu este utilizat. În interiorul acestei metode, clasa folosește potrivirea șabloanelor pentru a gestiona diferite scenarii bazate pe structura chunk-ului primit.

Apelarea `deep_symbolize_keys` asupra chunk-ului ajută la realizarea unei potriviri de șabloane mai elegante, permițându-ne să operăm cu simboluri în loc de șiruri de caractere.

```
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
```

Primul şablon pe care îl căutăm este un apel de instrument împreună cu numele funcţiei asociate. Dacă detectăm unul, îl introducem în array-ul `f_name`. Stocăm numele funcţiilor într-un array indexat, deoarece modelul este capabil să efectueze apeluri paralele ale funcţiilor, trimiţând mai multe funcţii spre executare în acelaşi timp.

> Apelarea paralelă a funcţiilor este capacitatea unui model AI de a efectua mai multe apeluri de funcţii împreună, permiţând ca efectele şi rezultatele acestor apeluri de funcţii să fie rezolvate în paralel. Acest lucru este deosebit de util dacă funcţiile durează mult timp şi reduce numărul de iteraţii cu API-ul, ceea ce la rândul său poate economisi o cantitate semnificativă de tokeni.

În continuare, trebuie să căutăm argumentele corespunzătoare apelurilor de funcţii.

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

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

Similar cu modul în care am gestionat numele funcţiilor, punem argumentele într-un array indexat.

În continuare, căutăm mesajele destinate utilizatorului, care vor sosi de la server câte un token odată şi vor fi atribuite variabilei `new_content`. De asemenea, trebuie să urmărim `finish_reason`. Acesta va fi `nil` până la ultimul fragment al secvenţei de ieşire.

```
  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
```

Important, adăugăm o expresie de potrivire a şabloanelor pentru a gestiona mesajele de

eroare trimise de furnizorul modelului AI. În mediile de dezvoltare locale, aruncăm o excepție, dar în producție, înregistrăm eroarea și finalizăm.

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

Clauza finală else a instrucțiunii case se va executa dacă niciunul dintre tiparele anterioare nu s-a potrivit. Este doar o măsură de siguranță, astfel încât să aflăm dacă modelul AI începe să ne trimită fragmente nerecunoscute.

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

Metoda `send_to_client` este responsabilă pentru trimiterea conținutului stocat în buffer către client. Aceasta verifică dacă buffer-ul nu este gol, actualizează conținutul mesajului bot, randează mesajul bot și salvează conținutul în baza de date pentru a asigura persistența datelor.

```ruby
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` este apelată când procesarea fluxului este completă. Aceasta execută apelurile de funcții dacă au fost primite în timpul fluxului, actualizează mesajul botului cu conținutul final și alte informații relevante și resetează istoricul apelurilor de funcții

```ruby
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
```

Dacă modelul decide să apeleze o funcție, trebuie să "dispecerizezi" acel apel de funcție (numele și argumentele) într-un mod în care să fie executat și mesajele `function_call` și `function_result` să fie adăugate la transcriptul conversației

Din experienţa mea, este mai bine să gestionezi crearea mesajelor de funcţii într-un singur loc în baza ta de cod, în loc să te bazezi pe implementările instrumentelor. Este mai curat, dar există şi un motiv practic foarte important: dacă modelul AI apelează o funcţie şi nu vede mesajele rezultate ale apelului şi rezultatului în transcript când faci o nouă iteraţie, *va apela aceeaşi funcţie din nou.* Potenţial la infinit. Ţine minte că AI-ul este complet fără stare, aşa că dacă nu îi transmiţi înapoi aceste apeluri de funcţii, ele nu s-au întâmplat.

```
# 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
```

Curăţarea istoricului apelurilor funcţiei după expediere este la fel de importantă ca şi asigurarea că apelul şi rezultatele ajung în transcriptul tău, astfel încât să nu continui să apelezi aceleaşi funcţii iar şi iar de fiecare dată când parcurgi bucla.

“Bucla de Conversaţie”

În clasa `PromptSubscriber`, folosim metoda `prompt` din modulul `PromptDeclarations` pentru a defini comportamentul buclei de conversaţie. Parametrul `until` este setat la `-> { bot_message.complete? }`, ceea ce înseamnă că bucla va continua până când `bot_message` este marcat ca fiind complet.

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

Dar când este `bot_message` marcat ca fiind complet? Dacă ați uitat, consultați din nou linia 13 din metoda `finalize`.

Să revizuim întreaga logică de procesare în flux.

1. `PromptSubscriber` primește un nou mesaj de la utilizator prin metoda `message_created`, care este invocată de sistemul pub/sub Wisper de fiecare dată când utilizatorul final creează un nou prompt.
2. Metoda de clasă `prompt` definește în mod declarativ comportamentul logicii de completare a chat-ului pentru `PromptSubscriber`. Modelul AI va executa o completare de chat cu conținutul mesajului utilizatorului, o nouă instanță de `ReplyStream` ca parametru de flux și condiția de buclă specificată.
3. Modelul AI procesează promptul și începe să genereze un răspuns. Pe măsură ce răspunsul este transmis în flux, metoda `call` a instanței `ReplyStream` este invocată pentru fiecare fragment de date.
4. Dacă modelul AI decide să apeleze o funcție utilitară, numele funcției și argumentele sunt extrase din fragment și stocate în matricele `f_name` și `f_arguments`.
5. Dacă modelul AI generează conținut vizibil utilizatorului, acesta este stocat temporar și trimis către client prin metoda `send_to_client`.
6. Odată ce procesarea fluxului este completă, metoda `finalize` este apelată. Dacă au fost invocate funcții utilitare în timpul fluxului, acestea sunt expediate folosind metoda `dispatch` a `PromptSubscriber`.
7. Metoda `dispatch` adaugă un mesaj `function_call` la transcriptul conversației, execută funcția utilitară corespunzătoare și adaugă un mesaj `function_result` la transcript cu rezultatul apelului funcției.

8. După expedierea funcțiilor utilitare, istoricul apelurilor de funcții este șters pentru a preveni apeluri duplicate de funcții în buclele ulterioare.
9. Dacă nu au fost invocate funcții utilitare, metoda `finalize` actualizează `bot_message` cu conținutul final, îl marchează ca fiind complet și trimite mesajul actualizat către client.
10. Condiția de buclă `-> { bot_message.complete? }` este evaluată. Dacă `bot_message` nu este marcat ca fiind complet, bucla continuă, iar promptul original este trimis din nou cu transcriptul conversației actualizat.
11. Pașii 3-10 sunt repetați până când `bot_message` este marcat ca fiind complet, indicând că modelul AI a terminat de generat răspunsul și nu mai sunt necesare funcții utilitare de executat.

Prin implementarea acestei bucle de conversație, permiteți modelului AI să se angajeze într-o interacțiune de tip du-te-vino cu aplicația, executând funcții utilitare după necesitate și generând răspunsuri vizibile utilizatorului până când conversația ajunge la o concluzie naturală.

Combinația dintre procesarea în flux și bucla de conversație permite experiențe dinamice și interactive bazate pe AI, unde modelul AI poate procesa input-ul utilizatorului, utiliza diverse unelte și funcții și oferi răspunsuri în timp real bazate pe contextul evoluat al conversației.

Continuare Automată

Este important să fim conștienți de limitările ieșirii AI. Majoritatea modelelor au un număr maxim de tokeni pe care îi pot genera într-un singur răspuns, care este determinat de parametrul `max_tokens`. Dacă modelul AI atinge această limită în timpul generării unui răspuns, se va opri brusc și va indica faptul că ieșirea a fost trunchiată.

În răspunsul transmis în flux de la API-ul platformei AI, puteți detecta această situație examinând variabila `finish_reason` din fragment. Dacă `finish_reason` este setat

la `"length"` (sau altă valoare cheie specifică modelului), înseamnă că modelul a atins limita maximă de tokeni în timpul generării și ieșirea a fost întreruptă.

O modalitate de a gestiona elegant acest scenariu și de a oferi o experiență fluidă utilizatorului este implementarea unui mecanism de continuare automată în logica de procesare a fluxului. Prin adăugarea unui pattern matching pentru motivele de finalizare legate de lungime, puteți alege să repetați bucla și să continuați automat ieșirea de unde s-a oprit.

Iată un exemplu intenționat simplificat despre cum puteți modifica metoda `call` din clasa `ReplyStream` pentru a suporta continuarea automată:

```
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
```

```
29    conversation.bot_message!(buffer.join, visible: false)
30    conversation.user_message!("please continue", visible: false)
31    bot_message.update_column(:created_at, Time.current)
32  end
```

În această versiune modificată, când `finish_reason` indică un output trunchiat, în loc să finalizăm fluxul, adăugăm o pereche de mesaje în transcript fără finalizare, mutăm mesajul original vizibil utilizatorului la "baza" transcriptului prin actualizarea atributului său `created_at`, și apoi lăsăm bucla să continue, astfel încât AI-ul să continue generarea de unde a rămas.

Țineți minte că endpoint-ul de completare AI este stateless. "Știe" doar ce îi comunicați prin transcript. În acest caz, modul în care comunicăm AI-ului că a fost întrerupt este prin adăugarea unor mesaje "invizibile" (pentru utilizatorul final) în transcript. Amintiți-vă totuși că acesta este un exemplu intenționat simplificat. O implementare reală ar necesita o gestionare suplimentară a transcriptului pentru a ne asigura că nu irosim token-i și/sau nu confuzăm AI-ul cu mesaje duplicate ale asistentului în transcript.

O implementare reală a auto-continuării ar trebui să aibă și așa-numita "logică de întrerupere de circuit" implementată pentru a preveni buclarea necontrolată. Motivul este că, date fiind anumite tipuri de prompt-uri ale utilizatorului și setări reduse ale `max_tokens`, AI-ul ar putea continua să genereze la nesfârșit output vizibil utilizatorului.

> Țineți cont că fiecare buclă necesită o cerere separată și că fiecare cerere consumă din nou întregul transcript. Ar trebui să luați în considerare cu atenție compromisurile dintre experiența utilizatorului și utilizarea API-ului atunci când decideți dacă să implementați auto-continuarea în aplicația dvs. Auto-continuarea poate fi în special periculos de costisitoare, mai ales când utilizați modele comerciale premium.

Concluzie

Procesarea în flux este un aspect critic în construirea aplicațiilor bazate pe AI care combină utilizarea de instrumente cu răspunsuri AI în timp real. Prin gestionarea eficientă a datelor în flux de la API-urile platformelor AI, puteți oferi o experiență de utilizare fluidă și interactivă, gestiona răspunsuri mari, optimiza utilizarea resurselor și trata elegant erorile.

Clasa `Conversation::ReplyStream` furnizată demonstrează cum procesarea în flux poate fi implementată într-o aplicație Ruby folosind potrivirea de șabloane și arhitectură bazată pe evenimente. Prin înțelegerea și valorificarea tehnicilor de procesare în flux, puteți debloca întregul potențial al integrării AI în aplicațiile dvs. și oferi experiențe de utilizare puternice și captivante.

Date cu Auto-vindecare

Datele cu auto-vindecare reprezintă o abordare puternică pentru asigurarea integrității, consistenței și calității datelor în aplicații prin valorificarea capacităților modelelor mari de limbaj (LLM). Această categorie de tipare se concentrează pe ideea de a utiliza IA pentru a detecta, diagnostica și corecta automat anomaliile, inconsistențele sau erorile din date, reducând astfel sarcina dezvoltatorilor și menținând un nivel ridicat de fiabilitate a datelor.

În esență, tiparele datelor cu auto-vindecare recunosc faptul că datele sunt elementul vital al oricărei aplicații, iar asigurarea acurateței și integrității acestora este crucială pentru funcționarea corectă și experiența utilizatorului în aplicație. Cu toate acestea, gestionarea și menținerea calității datelor poate fi o sarcină complexă și consumatoare de timp, mai ales pe măsură ce aplicațiile cresc în dimensiune și complexitate. Aici intervine puterea IA.

În tiparele datelor cu auto-vindecare, lucrătorii IA sunt folosiți pentru a monitoriza și analiza continuu datele aplicației tale. Aceste modele au capacitatea de a înțelege și interpreta tipare, relații și anomalii în cadrul datelor. Prin valorificarea capacităților lor de procesare și înțelegere a limbajului natural, pot identifica potențiale probleme sau inconsistențe în date și pot lua măsurile adecvate pentru a le rectifica.

Procesul de auto-vindecare a datelor implică de obicei mai mulți pași cheie:

1. **Monitorizarea Datelor**: Lucrătorii IA monitorizează constant fluxurile de date ale aplicației, bazele de date sau sistemele de stocare, căutând orice semne de anomalii, inconsistențe sau erori. Alternativ, poți activa o componentă IA ca reacție la o excepție.
2. **Detectarea Anomaliilor**: Când este detectată o problemă, lucrătorul IA analizează datele în detaliu pentru a identifica natura și scopul specific al problemei. Aceasta ar putea implica detectarea valorilor lipsă, a formatelor inconsistente sau a datelor care încalcă reguli sau constrângeri predefinite.
3. **Diagnosticare și Corecție**: Odată ce problema este identificată, lucrătorul IA își folosește cunoștințele și înțelegerea domeniului de date pentru a determina cursul adecvat de acțiune. Aceasta ar putea implica corectarea automată a datelor, completarea valorilor lipsă sau marcarea problemei pentru intervenție umană, dacă este necesar.
4. **Învățare Continuă (opțional, în funcție de caz)**: Pe măsură ce lucrătorul tău IA întâlnește și rezolvă diverse probleme de date, poate genera ieșiri care descriu ce s-a întâmplat și cum a răspuns. Aceste metadate pot fi introduse în procese de învățare care îți permit (și poate modelului subiacent, prin ajustare fină) să devii mai eficient și mai eficace în timp în identificarea și rezolvarea anomaliilor din date.

Prin detectarea și corectarea automată a problemelor de date, poți asigura că aplicația ta operează cu date de înaltă calitate și fiabile. Acest lucru reduce riscul ca erorile,

inconsistențele sau defectele legate de date să afecteze funcționalitatea sau experiența utilizatorului aplicației.

Odată ce ai lucrători IA care se ocupă de sarcina de monitorizare și corectare a datelor, îți poți concentra eforturile asupra altor aspecte critice ale aplicației. Acest lucru economisește timp și resurse care altfel ar fi cheltuite pe curățarea și întreținerea manuală a datelor. De fapt, pe măsură ce aplicațiile tale cresc în dimensiune și complexitate, gestionarea manuală a calității datelor devine din ce în ce mai dificilă. Tiparele "Date cu Auto-vindecare" se scalează eficient prin valorificarea puterii IA pentru a gestiona volume mari de date și a detecta probleme în timp real.

Datorită naturii lor, modelele IA se pot adapta la schimbările de tipare, scheme sau cerințe ale datelor în timp, cu puțină sau fără supraveghere. Atâta timp cât directivele lor oferă îndrumări adecvate, în special în ceea ce privește rezultatele intenționate, aplicația ta poate evolua și gestiona noi scenarii de date fără a necesita intervenție manuală extensivă sau modificări de cod.

Tiparele datelor cu auto-vindecare se aliniază bine cu celelalte categorii de tipare pe care le-am discutat, cum ar fi "Multitudinea de Lucrători". Capacitatea de auto-vindecare a datelor poate fi văzută ca un tip specializat de lucrător care se concentrează specific pe asigurarea calității și integrității datelor. Acest tip de lucrător operează alături de alți lucrători IA, fiecare contribuind la diferite aspecte ale funcționalității aplicației.

Implementarea tiparelor de date cu auto-vindecare în practică necesită o proiectare și integrare atentă a modelelor IA în arhitectura aplicației. Din cauza riscurilor de pierdere și corupere a datelor, ar trebui să definești linii directoare clare pentru modul în care vei utiliza această tehnică. Ar trebui, de asemenea, să iei în considerare factori precum performanța, scalabilitatea și securitatea datelor.

Studiu de Caz Practic: Repararea JSON-ului Defect

Una dintre cele mai practice și convenabile modalități de a valorifica datele cu auto-vindecare este și foarte simplă de explicat: repararea JSON-ului defect.

Această tehnică poate fi aplicată provocării comune de a gestiona date imperfecte sau inconsistente generate de LLM-uri, cum ar fi JSON-ul defect, și oferă o abordare pentru detectarea și corectarea automată a acestor probleme.

La Olympia întâlnesc în mod regulat scenarii în care modelele LLM generează date JSON care nu sunt perfect valide. Acest lucru se poate întâmpla din diverse motive, cum ar fi adăugarea de către LLM a unor comentarii înainte sau după codul JSON propriu-zis, sau introducerea unor erori de sintaxă precum virgule lipsă sau ghilimele duble neescapate. Aceste probleme pot duce la erori de parsare și pot cauza întreruperi în funcționalitatea aplicației.

Pentru a rezolva această problemă, am implementat o soluție practică sub forma unei clase JsonFixer. Această clasă încorporează modelul "Self-Healing Data" prin preluarea JSON-ului defect ca date de intrare și utilizarea unui LLM pentru a-l repara, păstrând în același timp cât mai multe informații și intenții posibile.

```
class JsonFixer
  include Raix::ChatCompletion

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

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

```
    transcript << { assistant: "Here is the corrected JSON\n```json\n" }

    self.stop = ["```"]

    chat_completion(json: true)
  end

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

Observați cum `JsonFixer` folosește Ventriloquist pentru a ghida răspunsurile AI-ului.

Procesul de auto-corectare a datelor JSON funcționează după cum urmează:

1. **Generarea JSON**: Un LLM este utilizat pentru a genera date JSON pe baza anumitor prompturi sau cerințe. Cu toate acestea, din cauza naturii LLM-urilor, JSON-ul generat nu este întotdeauna perfect valid. Parser-ul JSON va genera, desigur, o eroare `ParserError` dacă îi furnizați un JSON invalid.

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

Rețineți că mesajul de eroare este, de asemenea, transmis apelului `JSONFixer` astfel încât acesta să nu trebuiască să presupună complet ce este în neregulă cu datele, mai ales că parser-ul vă va spune adesea exact ce nu este în regulă.

2. **Corecția bazată pe LLM**: Clasa `JSONFixer` trimite JSON-ul defect înapoi către un LLM, împreună cu un prompt sau o instrucțiune specifică pentru a repara JSON-ul

păstrând cât mai mult posibil informațiile și intenția originală. LLM-ul, antrenat pe cantități vaste de date și cu o înțelegere a sintaxei JSON, încearcă să corecteze erorile și să genereze un șir JSON valid. Delimitarea Răspunsurilor este utilizată pentru a restricționa rezultatul LLM-ului, iar noi alegem Mixtral 8x7B ca model AI, deoarece este deosebit de bun pentru acest tip de sarcină.

3. **Validare și Integrare**: Șirul JSON reparat returnat de LLM este parsat de clasa `JSONFixer` însăși, deoarece a apelat `chat_completion(json: true)`. Dacă JSON-ul reparat trece de validare, este integrat înapoi în fluxul de lucru al aplicației, permițând aplicației să continue procesarea datelor fără probleme. JSON-ul defect a fost "vindecat".

Deși am scris și rescris propria mea implementare `JSONFixer` de mai multe ori, mă îndoiesc că timpul total investit în toate aceste versiuni este mai mult de o oră sau două.

Rețineți că păstrarea intenției este un element cheie al oricărui model de date cu auto-vindecare. Procesul de corecție bazat pe LLM urmărește să păstreze cât mai mult posibil informațiile și intenția originală a JSON-ului generat. Acest lucru asigură că JSON-ul reparat își menține semnificația semantică și poate fi utilizat eficient în contextul aplicației.

Această implementare practică a abordării "Date cu Auto-vindecare" în Olympia demonstrează clar cum AI-ul, în special LLM-urile, poate fi folosit pentru a rezolva provocări reale legate de date. Aceasta evidențiază puterea combinării tehnicilor tradiționale de programare cu capacitățile AI pentru a construi aplicații robuste și eficiente.

Legea lui Postel și Modelul "Date cu Auto-vindecare"

"Date cu Auto-vindecare", așa cum este exemplificat de clasa JSONFixer, se aliniază bine cu principiul cunoscut sub numele de Legea lui Postel, denumit și Principiul Robusteții. Legea lui Postel afirmă:

"Fii conservator în ceea ce faci, fii liberal în ceea ce accepți de la alții."

Acest principiu, articulat inițial de Jon Postel, un pionier al internetului timpuriu, subliniază importanța construirii unor sisteme care sunt tolerante la intrări diverse sau chiar ușor incorecte, menținând în același timp o aderență strictă la protocoalele specificate când se trimit ieșiri.

În contextul "Datelor cu Auto-vindecare", clasa JSONFixer întruchipează Legea lui Postel prin faptul că este liberală în acceptarea datelor JSON defecte sau imperfecte generate de LLM-uri. Nu respinge sau eșuează imediat când întâlnește JSON care nu aderă strict la formatul așteptat. În schimb, adoptă o abordare tolerantă și încearcă să repare JSON-ul folosind puterea LLM-urilor.

Prin faptul că este liberală în acceptarea JSON-ului imperfect, clasa JSONFixer demonstrează robustețe și flexibilitate. Aceasta recunoaște că datele din lumea reală vin adesea în diverse forme și pot să nu se conformeze întotdeauna specificațiilor stricte. Prin gestionarea și corectarea grațioasă a acestor abateri, clasa asigură că aplicația poate continua să funcționeze lin, chiar și în prezența datelor imperfecte.

Pe de altă parte, clasa JSONFixer aderă și la aspectul conservator al Legii lui Postel când vine vorba de ieșire. După repararea JSON-ului folosind LLM-uri, clasa validează JSON-ul corectat pentru a se asigura că se conformează strict formatului așteptat. Aceasta menține integritatea și corectitudinea datelor înainte de a le transmite către alte părți ale aplicației. Această abordare conservatoare garantează că ieșirea clasei JSONFixer este fiabilă și consistentă, promovând interoperabilitatea și prevenind propagarea erorilor.

Informații interesante despre Jon Postel:

- Jon Postel (1943-1998) a fost un informatician american care a jucat un rol

crucial în dezvoltarea Internetului. Era cunoscut drept "Zeul Internetului" pentru contribuțiile sale semnificative la protocoalele și standardele fundamentale.

- Postel a fost editorul seriei de documente Request for Comments (RFC), care este o serie de note tehnice și organizaționale despre Internet. A fost autor sau co-autor a peste 200 de RFC-uri, inclusiv protocoalele fundamentale precum TCP, IP și SMTP.
- Pe lângă contribuțiile sale tehnice, Postel era cunoscut pentru abordarea sa umilă și colaborativă. El credea în importanța ajungerii la un consens și a lucrului împreună pentru a construi o rețea robustă și interoperabilă.
- Postel a servit ca Director al Diviziei de Rețele de Calculatoare la Institutul de Științe Informaționale (ISI) al Universității din California de Sud (USC) din 1977 până la moartea sa prematură în 1998.
- În recunoașterea contribuțiilor sale imense, lui Postel i s-a acordat post-mortem prestigiosul Premiu Turing în 1998, adesea numit "Premiul Nobel pentru Informatică."

Clasa `JSONFixer` promovează robustețea, flexibilitatea și interoperabilitatea, care au fost valorile fundamentale pe care Postel le-a susținut de-a lungul carierei sale. Prin construirea unor sisteme care sunt tolerante la imperfecțiuni, menținând în același timp o aderență strictă la protocoale, putem crea aplicații care sunt mai reziliente și adaptabile în fața provocărilor din lumea reală.

Consideraţii şi Contraindicaţii

Aplicabilitatea abordărilor de date cu auto-remediere depinde în întregime de tipul de date pe care le gestionează aplicația dvs. Există un motiv pentru care s-ar putea să nu doriţi să modificaţi pur şi simplu `JSON.parse` pentru a corecta automat *toate erorile de parsare JSON* din aplicația dvs.: nu toate erorile pot sau ar trebui să fie corectate

automat.

Auto-remedierea este deosebit de complicată când este corelată cu cerințele de reglementare sau conformitate legate de manipularea și procesarea datelor. Unele industrii, precum cele din domeniul sănătății și finanțelor, au reglementări atât de stricte privind integritatea datelor și posibilitatea de auditare încât efectuarea oricărui tip de corecție a datelor de tip "cutie neagră" fără supraveghere sau înregistrare adecvată poate încălca aceste reglementări. Este crucial să vă asigurați că orice tehnică de auto-remediere a datelor pe care o dezvoltați se aliniază cu cadrele legale și de reglementare aplicabile.

Aplicarea tehnicilor de date cu auto-remediere, în special cele care implică modele AI, poate avea, de asemenea, un impact major asupra performanței aplicației și utilizării resurselor. Procesarea unor volume mari de date prin modele AI pentru detectarea și corectarea erorilor poate fi intensivă din punct de vedere computațional. Este important să evaluați compromisurile între beneficiile datelor cu auto-remediere și costurile asociate cu performanța și resursele.

Acestea fiind spuse, să explorăm factorii implicați în decizia când și unde să aplicăm această abordare puternică.

Criticitatea Datelor

Când se ia în considerare aplicarea tehnicilor de date cu auto-remediere, este crucial să evaluăm criticitatea datelor procesate. Nivelul de criticitate se referă la importanța și sensibilitatea datelor în contextul aplicației dvs. și al domeniului său de activitate.

În unele cazuri, corectarea automată a erorilor de date poate să nu fie adecvată, mai ales dacă datele sunt foarte sensibile sau au implicații legale. De exemplu, luați în considerare următoarele scenarii:

1. **Tranzacții Financiare**: În aplicațiile financiare, cum ar fi sistemele bancare sau platformele de tranzacționare, acuratețea datelor este de maximă importanță.

Chiar și erorile minore în datele financiare pot avea consecințe semnificative, cum ar fi solduri incorecte ale conturilor, fonduri direcționate greșit sau decizii eronate de tranzacționare. În aceste cazuri, corecțiile automatizate fără verificare și auditare temeinică pot introduce riscuri inacceptabile.

2. **Fișe Medicale**: Aplicațiile din domeniul sănătății gestionează date ale pacienților foarte sensibile și confidențiale. Inexactitățile din fișele medicale pot avea implicații severe pentru siguranța pacientului și deciziile de tratament. Modificarea automată a datelor medicale fără supraveghere și validare corespunzătoare din partea profesioniștilor din domeniul sănătății poate încălca cerințele de reglementare și pune în pericol bunăstarea pacientului.
3. **Documente Juridice**: Aplicațiile care gestionează documente juridice, cum ar fi contracte, acorduri sau dosare judecătorești, necesită acuratețe și integritate strictă. Chiar și erorile minore în datele juridice pot avea ramificații legale semnificative. Corecțiile automatizate în acest domeniu pot să nu fie adecvate, deoarece datele necesită adesea revizuire manuală și verificare de către experți juridici pentru a asigura validitatea și aplicabilitatea acestora.

În aceste scenarii cu date critice, riscurile asociate cu corecțiile automatizate depășesc adesea beneficiile potențiale. Consecințele introducerii erorilor sau modificării incorecte a datelor pot fi severe, ducând la pierderi financiare, răspunderi juridice sau chiar vătămarea persoanelor.

Când se lucrează cu date foarte critice, este esențial să se acorde prioritate proceselor de verificare și validare manuală. Supravegherea și expertiza umană sunt cruciale în asigurarea acurateței și integrității datelor. Tehnicile automatizate de auto-remediere pot fi încă utilizate pentru a semnala potențiale erori sau inconsistențe, dar decizia finală privind corecțiile ar trebui să implice judecată și aprobare umană.

Cu toate acestea, este important de menționat că nu toate datele dintr-o aplicație pot avea același nivel de criticitate. În cadrul aceleiași aplicații, pot exista subseturi de date care sunt mai puțin sensibile sau au un impact mai redus în cazul apariției erorilor.

În astfel de cazuri, tehnicile de date cu auto-remediere pot fi aplicate selectiv acelor subseturi specifice de date, în timp ce datele critice rămân supuse verificării manuale.

Cheia este să evaluați cu atenție criticitatea fiecărei categorii de date din aplicația dvs. și să definiți orientări și procese clare pentru gestionarea corecțiilor bazate pe riscurile și implicațiile asociate. Prin diferențierea între datele critice (de exemplu, registre contabile, fișe medicale) și datele non-critice (de exemplu, adrese poștale, avertismente privind resursele), puteți găsi un echilibru între valorificarea beneficiilor tehnicilor de date cu auto-remediere acolo unde este adecvat și menținerea unui control și a unei supravegheri stricte unde este necesar.

În cele din urmă, decizia de a aplica tehnici de date cu auto-remediere pentru date critice ar trebui luată în consultare cu experți în domeniu, consilieri juridici și alte părți interesate relevante. Este esențial să se ia în considerare cerințele specifice, reglementările și riscurile asociate cu datele aplicației dvs. și să se alinieze strategiile de corecție a datelor în consecință.

Severitatea Erorilor

Când se aplică tehnici de date cu auto-remediere, este important să se evalueze severitatea și impactul erorilor de date. Nu toate erorile sunt create egale, iar acțiunea adecvată poate varia în funcție de severitatea problemei.

Inconsistențele minore sau problemele de formatare pot fi potrivite pentru corecție automată. De exemplu, un worker de date cu auto-remediere însărcinat cu repararea JSON-ului defect poate gestiona virgule lipsă sau ghilimele duble neescapate fără a modifica semnificativ sensul sau structura datelor. Aceste tipuri de erori sunt adesea simple de corectat și au un impact minim asupra integrității generale a datelor.

Cu toate acestea, erorile mai severe care modifică fundamental sensul sau integritatea datelor pot necesita o abordare diferită. În astfel de cazuri, corecțiile automatizate pot să nu fie suficiente, iar intervenția umană poate fi necesară pentru a asigura acuratețea și validitatea datelor.

Aici intervine conceptul de utilizare a AI-ului în sine pentru a ajuta la determinarea severității erorilor. Prin valorificarea capacităților modelelor de AI, putem proiecta lucrători de date cu auto-remediere care nu doar corectează erorile, ci și evaluează severitatea acestora și iau decizii informate cu privire la modul de gestionare a lor.

De exemplu, să luăm în considerare un lucrător de date cu auto-remediere responsabil pentru corectarea inconsistențelor în datele care intră într-o bază de date a clienților. Lucrătorul poate fi proiectat să analizeze datele și să identifice potențiale erori, cum ar fi informațiile lipsă sau contradictorii. Cu toate acestea, în loc să corecteze automat toate erorile, lucrătorul poate fi echipat cu apeluri de instrumente suplimentare care îi permit să marcheze erorile severe pentru revizuire umană.

Iată un exemplu despre cum poate fi implementat acest lucru:

```
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
```

În acest exemplu, lucrătorul `CustomerDataHealer` este proiectat pentru a identifica și corecta inconsistențele din datele clienților. Din nou, folosim Response Fencing și Ventriloquist pentru a obține ieșiri structurate. Este important de menționat că directiva de sistem a lucrătorului include instrucțiuni pentru utilizarea funcției `flag_for_review` în cazul în care sunt întâlnite erori grave.

Când lucrătorul procesează datele clienților, acesta analizează datele și încearcă să corecteze orice inconsistențe. Dacă lucrătorul determină că erorile sunt grave și necesită intervenție umană, poate utiliza instrumentul `flag_for_review` pentru a marca datele și a furniza un motiv pentru marcare.

Metoda `chat_completion` este apelată cu `json: true` pentru a analiza datele corectate ale clientului ca JSON. Nu există nicio prevedere pentru buclare după un apel de funcție, astfel că rezultatul va fi gol dacă `flag_for_review` a fost invocat. În caz contrar, clientul este actualizat cu datele revizuite și potențial corectate.

Prin încorporarea evaluării severității erorilor și opțiunea de a marca datele pentru revizuire umană, lucrătorul de date cu auto-vindecare devine mai inteligent și adaptabil. Poate gestiona erori minore în mod automat, în timp ce escaladează erorile grave către experți umani pentru intervenție manuală.

Criteriile specifice pentru determinarea severității erorilor pot fi definite în directiva lucrătorului pe baza cunoștințelor domeniului și a cerințelor de afaceri. Factori precum

impactul asupra integrității datelor, potențialul de pierdere sau corupere a datelor și consecințele datelor incorecte pot fi luați în considerare la evaluarea severității.

Prin utilizarea AI-ului pentru evaluarea severității erorilor și oferirea opțiunilor pentru intervenție umană, tehnicile de date cu auto-vindecare pot atinge un echilibru între automatizare și menținerea acurateței datelor. Această abordare asigură că erorile minore sunt corectate eficient, în timp ce erorile grave primesc atenția și expertiza necesară din partea revizorilor umani.

Complexitatea Domeniului

Când se ia în considerare aplicarea tehnicilor de date cu auto-vindecare, este important să se evalueze complexitatea domeniului de date și regulile care guvernează structura și relațiile acestuia. Complexitatea domeniului poate impacta semnificativ eficacitatea și fezabilitatea abordărilor de corectare automată a datelor.

Tehnicile de date cu auto-vindecare funcționează bine când datele urmează tipare și constrângeri bine definite. În domeniile unde structura datelor este relativ simplă și relațiile dintre elementele de date sunt directe, corecțiile automate pot fi aplicate cu un grad ridicat de încredere. De exemplu, corectarea problemelor de formatare sau impunerea constrângerilor de tip de date de bază pot fi gestionate eficient de lucrătorii de date cu auto-vindecare.

Cu toate acestea, pe măsură ce complexitatea domeniului de date crește, provocările asociate cu corectarea automată a datelor cresc și ele. În domeniile cu logică de afaceri complexă, relații complexe între entitățile de date sau reguli și excepții specifice domeniului, tehnicile de date cu auto-vindecare pot să nu capteze întotdeauna nuanțele și pot introduce consecințe nedorite.

Să luăm în considerare un exemplu de domeniu complex: un sistem de tranzacționare financiară. În acest domeniu, datele implică diverse instrumente financiare, date de piață, reguli de tranzacționare și cerințe de reglementare. Relațiile dintre diferitele

elemente de date pot fi complexe, iar regulile care guvernează validitatea și consistența datelor pot fi foarte specifice domeniului.

Într-un astfel de domeniu complex, un lucrător de date cu auto-vindecare însărcinat cu corectarea inconsistențelor în datele de tranzacționare ar trebui să aibă o înțelegere profundă a regulilor și constrângerilor specifice domeniului. Ar trebui să ia în considerare factori precum reglementările pieței, limitele de tranzacționare, calculele de risc și procedurile de decontare. Corecțiile automate în acest context pot să nu capteze întotdeauna complexitatea completă a domeniului și pot introduce accidental erori sau pot încălca reguli specifice domeniului.

Pentru a aborda provocările complexității domeniului, tehnicile de date cu auto-vindecare pot fi îmbunătățite prin încorporarea cunoștințelor și regulilor specifice domeniului în modelele și lucrătorii AI. Acest lucru poate fi realizat prin tehnici precum:

1. **Instruire Specifică Domeniului**: Modelele AI utilizate pentru date cu auto-vindecare pot fi direcționate sau chiar ajustate fin pe seturi de date specifice domeniului care captează complexitățile și regulile domeniului particular. Prin expunerea modelelor la date și scenarii reprezentative, acestea pot învăța tiparele, constrângerile și excepțiile specifice domeniului.
2. **Constrângeri Bazate pe Reguli**: Lucrătorii de date cu auto-vindecare pot fi augmentați cu constrângeri explicite bazate pe reguli care codifică cunoștințele specifice domeniului. Aceste reguli pot fi definite de experți în domeniu și integrate în procesul de corectare a datelor. Modelele AI pot apoi să utilizeze aceste reguli pentru a-și ghida deciziile și a asigura conformitatea cu cerințele specifice domeniului.
3. **Colaborarea cu Experții în Domeniu**: În domeniile complexe, este crucial să implici experți în domeniu în proiectarea și dezvoltarea tehnicilor de date cu auto-vindecare. Experții în domeniu pot oferi perspective valoroase asupra complexităților datelor, regulilor de afaceri și potențialelor cazuri limită. Cunoștințele lor pot fi încorporate în modelele și lucrătorii AI pentru a îmbunătăți

acurateţea şi fiabilitatea corecţiilor automate de date folosind modele Human In The Loop.

4. **Abordare Incrementală şi Iterativă**: Când se lucrează cu domenii complexe, este adesea benefic să se adopte o abordare incrementală şi iterativă pentru datele cu auto-vindecare. În loc să se încerce automatizarea corecţiilor pentru întregul domeniu dintr-o dată, concentrează-te pe subdomenii specifice sau categorii de date unde regulile şi constrângerile sunt bine înţelese. Extinde treptat scopul tehnicilor de auto-vindecare pe măsură ce înţelegerea domeniului creşte şi tehnicile se dovedesc eficiente.

Luând în considerare complexitatea domeniului datelor şi încorporând cunoştinţele specifice domeniului în tehnicile de date cu auto-remediere, puteţi găsi un echilibru între automatizare şi acurateţe. Este important să recunoaştem că datele cu auto-remediere nu reprezintă o soluţie universal valabilă şi că abordarea ar trebui adaptată cerinţelor şi provocărilor specifice fiecărui domeniu.

În domeniile complexe, o abordare hibridă care combină tehnicile de date cu auto-remediere cu expertiza şi supravegherea umană poate fi cea mai eficientă. Corecţiile automatizate pot gestiona cazurile de rutină şi bine definite, în timp ce scenariile complexe sau excepţiile pot fi marcate pentru revizuire şi intervenţie umană. Această abordare colaborativă asigură că beneficiile automatizării sunt realizate, menţinând în acelaşi timp controlul şi acurateţea necesare în domeniile de date complexe.

Explicabilitate şi Transparenţă

Explicabilitatea se referă la capacitatea de a înţelege şi interpreta raţionamentul din spatele deciziilor luate de modelele AI, în timp ce transparenţa implică oferirea unei vizibilităţi clare asupra procesului de corectare a datelor.

În multe contexte, modificările datelor trebuie să fie auditabile şi justificabile. Părţile interesate, inclusiv utilizatorii de business, auditorii şi organismele de reglementare, pot

solicita explicații pentru anumite corecții ale datelor și modul în care modelele AI au ajuns la aceste decizii. Acest lucru este crucial mai ales în domeniile unde acuratețea și integritatea datelor au implicații semnificative, cum ar fi finanțele, sănătatea și aspectele juridice.

Pentru a aborda necesitatea explicabilității și transparenței, tehnicile de date cu auto-remediere ar trebui să încorporeze mecanisme care oferă perspective asupra procesului decizional al modelelor AI. Acest lucru poate fi realizat prin diverse abordări:

1. **Lanțul de Gândire**: Solicitarea modelului să își explice gândirea "cu voce tare" înainte de a aplica modificări datelor poate permite o înțelegere mai ușoară a procesului decizional și poate genera explicații ușor de citit pentru corecțiile făcute. Compromisul este o complexitate puțin mai mare în separarea explicației de ieșirea datelor structurate, care poate fi abordată prin...
2. **Generarea Explicațiilor**: Lucrătorii de date cu auto-remediere pot fi dotați cu capacitatea de a genera explicații ușor de înțeles pentru corecțiile pe care le fac. Acest lucru poate fi realizat prin solicitarea modelului să producă procesul său decizional sub formă de explicații ușor de înțeles *integrate în datele însele*. De exemplu, un lucrător de date cu auto-remediere ar putea genera un raport care evidențiază inconsistențele specifice ale datelor identificate, corecțiile aplicate și raționamentul din spatele acestor corecții.
3. **Importanța Caracteristicilor**: Modelele AI pot fi instruite cu informații despre importanța diferitelor caracteristici sau atribute în procesul de corectare a datelor ca parte a directivelor lor. Aceste directive, la rândul lor, pot fi expuse părților interesate umane. Prin identificarea factorilor cheie care influențează deciziile modelului, părțile interesate pot obține perspective asupra raționamentului din spatele corecțiilor și pot evalua validitatea acestora.
4. **Înregistrare și Auditare**: Implementarea unor mecanisme comprehensive de înregistrare și auditare este crucială pentru menținerea transparenței în procesul de date cu auto-remediere. Fiecare corecție a datelor făcută de modelele AI

ar trebui înregistrată, incluzând datele originale, datele corectate şi acţiunile specifice întreprinse. Această pistă de audit permite analiza retrospectivă şi oferă o înregistrare clară a modificărilor făcute datelor.

5. **Abordarea cu Omul în Bucla de Decizie**: Încorporarea unei abordări cu omul în bucla de decizie poate îmbunătăţi explicabilitatea şi transparenţa tehnicilor de date cu auto-remediere. Prin implicarea experţilor umani în revizuirea şi validarea corecţiilor generate de AI, organizaţiile pot asigura că corecţiile se aliniază cu cunoştinţele domeniului şi cerinţele de business. Supravegherea umană adaugă un strat suplimentar de responsabilitate şi permite identificarea oricăror potenţiale prejudecăţi sau erori în modelele AI.
6. **Monitorizare şi Evaluare Continuă**: Monitorizarea şi evaluarea regulată a performanţei tehnicilor de date cu auto-remediere este esenţială pentru menţinerea transparenţei şi încrederii. Prin evaluarea acurateţei şi eficacităţii modelelor AI în timp, organizaţiile pot identifica orice abateri sau anomalii şi pot lua măsuri corective. Monitorizarea continuă ajută la asigurarea că procesul de date cu auto-remediere rămâne fiabil şi aliniat cu rezultatele dorite.

Explicabilitatea şi transparenţa sunt consideraţii critice în implementarea tehnicilor de date cu auto-remediere. Prin oferirea de explicaţii clare pentru corecţiile datelor, menţinerea unor piste de audit comprehensive şi implicarea supravegherii umane, organizaţiile pot construi încredere în procesul de date cu auto-remediere şi pot asigura că modificările făcute datelor sunt justificabile şi aliniate cu obiectivele de business.

Este important să se găsească un echilibru între beneficiile automatizării şi necesitatea transparenţei. În timp ce tehnicile de date cu auto-remediere pot îmbunătăţi semnificativ calitatea datelor şi eficienţa, acestea nu ar trebui să vină cu costul pierderii vizibilităţii şi controlului asupra procesului de corectare a datelor. Prin proiectarea lucrătorilor de date cu auto-remediere având în vedere explicabilitatea şi transparenţa, organizaţiile pot valorifica puterea AI menţinând în acelaşi timp nivelul necesar de responsabilitate şi încredere în date.

Consecinţe Neintenţionate

În timp ce tehnicile de date cu auto-remediere urmăresc să îmbunătățească calitatea și consistența datelor, este crucial să fim conştienţi de potențialul consecințelor neintenționate. Corecțiile automatizate, dacă nu sunt proiectate şi monitorizate cu atenție, pot modifica neintenționat sensul sau contextul datelor, ducând la probleme în aval.

Unul dintre riscurile principale ale datelor cu auto-remediere este introducerea prejudecăților sau erorilor în procesul de corectare a datelor. Modelele AI, ca orice alt sistem software, pot fi supuse prejudecăților prezente în datele de antrenament sau introduse prin proiectarea algoritmilor. Dacă aceste prejudecăți nu sunt identificate și atenuate, ele se pot propaga prin procesul de date cu auto-remediere și pot rezulta în modificări de date distorsionate sau incorecte.

De exemplu, să luăm în considerare un operator de date cu auto-vindecare însărcinat cu corectarea inconsistențelor în datele demografice ale clienților. Dacă modelul de IA a învățat prejudecăți din datele istorice, cum ar fi asocierea anumitor ocupații sau niveluri de venit cu genuri sau etnii specifice, acesta ar putea face presupuneri incorecte și ar putea modifica datele într-un mod care întărește aceste prejudecăți. Acest lucru poate duce la profile inexacte ale clienților, decizii de afaceri greșite și potențiale rezultate discriminatorii.

O altă consecință neintențională potențială este pierderea informațiilor sau contextului valoros în timpul procesului de corectare a datelor. Tehnicile de date cu auto-vindecare se concentrează adesea pe standardizarea și normalizarea datelor pentru a asigura consistența. Cu toate acestea, în unele cazuri, datele originale pot conține nuanțe, excepții sau informații contextuale care sunt importante pentru înțelegerea imaginii complete. Corecțiile automatizate care aplică standardizarea în mod orb pot elimina sau ascunde involuntar aceste informații valoroase.

De exemplu, imaginați-vă un operator de date cu auto-vindecare responsabil pentru

corectarea inconsistențelor în fișele medicale. Dacă operatorul întâlnește istoricul medical al unui pacient cu o afecțiune rară sau un plan de tratament neobișnuit, ar putea încerca să normalizeze datele pentru a se potrivi unui model mai comun. Cu toate acestea, făcând acest lucru, ar putea pierde detaliile specifice și contextul care sunt cruciale pentru reprezentarea exactă a situației unice a pacientului. Această pierdere de informații poate avea implicații serioase pentru îngrijirea pacientului și luarea deciziilor medicale.

Pentru a reduce riscurile consecințelor neintenționate, este esențial să adoptăm o abordare proactivă în proiectarea și implementarea tehnicilor de date cu auto-vindecare:

1. **Testare și Validare Minuțioasă**: Înainte de a implementa operatori de date cu auto-vindecare în producție, este crucial să testăm și să validăm minuțios comportamentul acestora în diverse scenarii. Aceasta include testarea cu seturi de date reprezentative care acoperă diverse cazuri limită, excepții și potențiale prejudecăți. Testarea riguroasă ajută la identificarea și abordarea oricăror consecințe neintenționate înainte ca acestea să afecteze datele din lumea reală.
2. **Monitorizare și Evaluare Continuă**: Implementarea mecanismelor de monitorizare și evaluare continuă este esențială pentru detectarea și atenuarea consecințelor neintenționate în timp. Revizuirea regulată a rezultatelor proceselor de date cu auto-vindecare, analizarea impactului asupra sistemelor din aval și a procesului decizional, precum și colectarea feedback-ului de la părțile interesate pot ajuta la identificarea oricăror efecte adverse și pot declanșa acțiuni corective în timp util. Dacă organizația dvs. are tablouri de bord operaționale, adăugarea unor metrici clar vizibile legate de modificările automate ale datelor este probabil o idee bună. Adăugarea alarmelor conectate la deviații mari de la activitatea normală de modificare a datelor este probabil o idee și mai bună!
3. **Supraveghere și Intervenție Umană**: Menținerea supravegherii umane și a capacității de a interveni în procesul de date cu auto-vindecare este crucială. În timp ce automatizarea poate îmbunătăți semnificativ eficiența, este important să

existe experți umani care să revizuiască și să valideze corecțiile făcute de modelele de IA, în special în domeniile critice sau sensibile. Judecata umană și expertiza în domeniu pot ajuta la identificarea și abordarea oricăror consecințe neintenționate care pot apărea.

4. **IA Explicabilă (XAI) și Transparență**: După cum s-a discutat în subsecțiunea anterioară, încorporarea tehnicilor de IA explicabilă și asigurarea transparenței în procesul de date cu auto-vindecare pot ajuta la atenuarea consecințelor neintenționate. Prin furnizarea de explicații clare pentru corecțiile datelor și menținerea unor piste de audit cuprinzătoare, organizațiile pot înțelege mai bine și pot urmări raționamentul din spatele modificărilor făcute de modelele de IA.
5. **Abordare Incrementală și Iterativă**: Adoptarea unei abordări incrementale și iterative pentru datele cu auto-vindecare poate ajuta la minimizarea riscului de consecințe neintenționate. În loc să aplicați corecții automatizate întregului set de date dintr-o dată, începeți cu un subset de date și extindeți treptat domeniul de aplicare pe măsură ce tehnicile se dovedesc eficiente și fiabile. Acest lucru permite monitorizarea atentă și ajustarea pe parcurs, reducând impactul oricăror consecințe neintenționate.
6. **Colaborare și Feedback**: Implicarea părților interesate din diferite domenii și încurajarea colaborării și feedback-ului pe tot parcursul procesului de date cu auto-vindecare poate ajuta la identificarea și abordarea consecințelor neintenționate. Solicitarea regulată a contribuției experților în domeniu, a consumatorilor de date și a utilizatorilor finali poate oferi perspective valoroase asupra impactului real al corecțiilor de date și poate evidenția orice probleme care ar fi putut fi trecute cu vederea.

Prin abordarea proactivă a riscului de consecințe neintenționate și implementarea măsurilor de protecție adecvate, organizațiile pot valorifica beneficiile tehnicilor de date cu auto-vindecare, minimizând în același timp potențialele efecte adverse. Este important să abordăm datele cu auto-vindecare ca pe un proces iterativ și colaborativ,

monitorizând, evaluând și rafinând continuu tehnicile pentru a ne asigura că se aliniază cu rezultatele dorite și mențin integritatea și fiabilitatea datelor.

Când luăm în considerare utilizarea modelelor de date cu auto-vindecare, este esențial să evaluăm cu atenție acești factori și să cântărim beneficiile în raport cu potențialele riscuri și limitări. În unele cazuri, o abordare hibridă care combină corecțiile automatizate cu supravegherea și intervenția umană poate fi soluția cea mai potrivită.

De asemenea, merită menționat că tehnicile de date cu auto-vindecare nu ar trebui văzute ca un înlocuitor pentru validarea robustă a datelor, sanitizarea datelor de intrare și mecanismele de gestionare a erorilor. Aceste practici fundamentale rămân critice pentru asigurarea integrității și securității datelor. Datele cu auto-vindecare ar trebui văzute ca o abordare complementară care poate augmenta și îmbunătăți aceste măsuri existente.

În cele din urmă, decizia de a utiliza modele de date cu auto-vindecare depinde de cerințele specifice, constrângerile și prioritățile aplicației dumneavoastră. Prin analizarea atentă a considerațiilor prezentate mai sus și alinierea acestora cu obiectivele și arhitectura aplicației dumneavoastră, puteți lua decizii informate cu privire la când și cum să utilizați eficient tehnicile de date cu auto-vindecare.

Generarea de Conţinut Contextual

Tiparele de Generare de Conţinut Contextual valorifică puterea modelelor lingvistice mari (MLM) pentru a genera conţinut dinamic şi specific contextului în cadrul aplicaţiilor. Această categorie de tipare recunoaşte importanţa furnizării de conţinut personalizat şi relevant pentru utilizatori, bazat pe nevoile lor specifice, preferinţe şi chiar pe interacţiunile anterioare şi actuale cu aplicaţia.

> În contextul acestei abordări, “conţinutul” se referă atât la conţinutul primar (adică postări pe blog, articole etc.), cât şi la meta-conţinut, cum ar fi recomandările pentru conţinutul primar.

Tiparele de Generare de Conținut Contextual pot juca un rol crucial în îmbunătățirea nivelurilor de implicare ale utilizatorilor, oferind experiențe personalizate și automatizând sarcinile de creare a conținutului atât pentru tine, cât și pentru utilizatorii tăi. Prin utilizarea tiparelor pe care le descriem în acest capitol, poți crea aplicații care generează conținut în mod dinamic, adaptându-se la context și la date de intrare în timp real.

Tiparele funcționează prin integrarea MLM-urilor în rezultatele aplicației, de la interfața utilizator (uneori denumită "chrome"), până la e-mailuri și alte forme de notificări, precum și orice fluxuri de generare a conținutului.

Când un utilizator interacționează cu aplicația sau declanșează o cerere specifică de conținut, aplicația captează contextul relevant, cum ar fi preferințele utilizatorului, interacțiunile anterioare sau prompturi specifice. Aceste informații contextuale sunt apoi transmise către MLM, împreună cu orice șabloane sau ghiduri necesare și sunt folosite pentru a produce text care altfel ar fi trebuit să fie fie codat direct, stocat într-o bază de date sau generat algoritmic.

Conținutul generat de MLM poate lua diverse forme, cum ar fi recomandări personalizate, descrieri dinamice ale produselor, răspunsuri personalizate prin e-mail sau chiar articole sau postări de blog complete. Una dintre cele mai radicale utilizări ale acestui conținut pe care am pionerat-o acum mai bine de un an este generarea dinamică a elementelor interfeței utilizator, cum ar fi etichetele formularelor, indiciile și alte tipuri de text explicativ.

Personalizare

Unul dintre beneficiile cheie ale tiparelor de Generare de Conținut Contextual este capacitatea de a oferi experiențe extrem de personalizate utilizatorilor. Prin generarea de conținut bazat pe contextul specific utilizatorului, aceste tipare permit aplicațiilor să adapteze conținutul la interesele, preferințele și interacțiunile individuale ale utilizatorilor.

Personalizarea merge dincolo de simpla inserare a numelui unui utilizator în conținut generic. Implică valorificarea contextului bogat disponibil despre fiecare utilizator pentru a genera conținut care rezonează cu nevoile și dorințele lor specifice. Acest context poate include o gamă largă de factori, cum ar fi:

1. **Informații despre Profilul Utilizatorului**: La cel mai general nivel de aplicare a acestei tehnici, datele demografice, interesele, preferințele și alte atribute ale profilului pot fi utilizate pentru a genera conținut care se aliniază cu experiența și caracteristicile utilizatorului.
2. **Date Comportamentale**: Interacțiunile anterioare ale unui utilizator cu aplicația, cum ar fi paginile vizualizate, linkurile accesate sau produsele cumpărate, pot oferi informații valoroase despre comportamentul și interesele lor. Aceste date pot fi utilizate pentru a genera sugestii de conținut care reflectă modelele lor de implicare și prezic nevoile viitoare.
3. **Factori Contextuali**: Contextul actual al utilizatorului, cum ar fi locația, dispozitivul, ora zilei sau chiar vremea, poate influența procesul de generare a conținutului. De exemplu, o aplicație de călătorii ar putea avea un worker AI care să fie capabil să genereze recomandări personalizate bazate pe locația curentă a utilizatorului și condițiile meteorologice predominante.

Prin valorificarea acestor factori contextuali, tiparele de Generare de Conținut Contextual permit aplicațiilor să livreze conținut care pare făcut special pentru fiecare utilizator în parte. Acest nivel de personalizare are mai multe beneficii semnificative:

1. **Creșterea Implicării**: Conținutul personalizat captează atenția utilizatorilor și îi menține implicați în aplicație. Când utilizatorii simt că conținutul este relevant și le vorbește direct despre nevoile lor, sunt mai predispuși să petreacă mai mult timp interacționând cu aplicația și explorând funcționalitățile acesteia.
2. **Îmbunătățirea Satisfacției Utilizatorilor**: Conținutul personalizat demonstrează că aplicația înțelege și se preocupă de cerințele unice ale utilizatorului. Prin

furnizarea de conținut care este util, informativ și aliniat cu interesele lor, aplicația poate spori satisfacția utilizatorilor și poate construi o conexiune mai puternică cu aceștia.

3. **Rate de Conversie Mai Mari**: În contextul aplicațiilor de e-commerce sau marketing, conținutul personalizat poate avea un impact semnificativ asupra ratelor de conversie. Prin prezentarea către utilizatori a unor produse, oferte sau recomandări care sunt adaptate preferințelor și comportamentului lor, aplicația poate crește probabilitatea ca utilizatorii să întreprindă acțiunile dorite, cum ar fi efectuarea unei achiziții sau înscrierea la un serviciu.

Productivitate

Tiparele de Generare de Conținut Contextual pot crește semnificativ anumite tipuri de productivitate prin reducerea necesității de generare și editare manuală a conținutului în procesele creative. Prin valorificarea puterii MLM-urilor, poți genera conținut de înaltă calitate la scară, economisind timp și efort pe care creatorii tăi de conținut și dezvoltatorii ar trebui altfel să îl petreacă făcând muncă manuală plictisitoare.

În mod tradițional, creatorii de conținut trebuie să cerceteze, să scrie, să editeze și să formateze conținutul pentru a se asigura că acesta îndeplinește cerințele aplicației și așteptările utilizatorilor. Acest proces poate fi consumator de timp și resurse, mai ales pe măsură ce volumul de conținut crește.

Cu toate acestea, folosind modele de Generare Contextuală de Conținut, procesul de creare a conținutului poate fi în mare parte automatizat. LLM-urile pot genera conținut coerent, corect din punct de vedere gramatical și relevant contextual, bazat pe prompturile și instrucțiunile furnizate. Această automatizare oferă mai multe beneficii de productivitate:

1. **Efort Manual Redus**: Prin delegarea sarcinilor de generare a conținutului către LLM-uri, creatorii de conținut se pot concentra pe sarcini de nivel superior,

precum strategia de conținut, ideația și asigurarea calității. Ei pot furniza contextul necesar, șabloanele și instrucțiunile către LLM și pot lăsa acestuia sarcina efectivă de generare a conținutului. Acest lucru reduce efortul manual necesar pentru scriere și editare, permițând creatorilor de conținut să fie mai productivi și eficienți.

2. **Creare Mai Rapidă de Conținut**: LLM-urile pot genera conținut mult mai rapid decât scriitorii umani. Cu prompturile și instrucțiunile potrivite, un LLM poate produce multiple bucăți de conținut în câteva secunde sau minute. Această viteză permite aplicațiilor să genereze conținut într-un ritm mult mai alert, ținând pasul cu cerințele utilizatorilor și cu peisajul digital în continuă schimbare.

> Oare crearea mai rapidă de conținut duce la o situație de "tragedia bunurilor comune" în care internetul este inundat de conținut pe care nimeni nu-l citește? Din păcate, suspectez că răspunsul este da.

3. **Consistență și Calitate**: LLM-urile pot revizui cu ușurință conținutul astfel încât să fie consistent în stil, ton și calitate. Cu instrucțiuni și exemple clare, anumite tipuri de aplicații (de exemplu, redacții, PR etc.) pot asigura că conținutul generat de oameni se aliniază cu vocea brandului lor și îndeplinește standardele de calitate dorite. Această consistență reduce nevoia de editare și revizuiri extensive, economisind timp și efort în procesul de creare a conținutului.
4. **Iterație și Optimizare**: Modelele de Generare Contextuală de Conținut permit iterația și optimizarea rapidă a conținutului. Prin ajustarea prompturilor, șabloanelor sau instrucțiunilor furnizate LLM-ului, aplicațiile dvs. pot genera rapid variații ale conținutului și testa diferite abordări într-un mod automatizat care nu a fost niciodată posibil în trecut. Acest proces iterativ permite experimentarea și rafinarea mai rapidă a strategiilor de conținut, ducând la un

conținut mai eficient și mai angajant în timp. Această tehnică particulară poate fi o adevărată revoluție pentru aplicații precum comerțul electronic care trăiesc și mor în funcție de ratele de abandon și angajament

Este important de menționat că, deși modelele de Generare Contextuală de Conținut pot îmbunătăți semnificativ productivitatea, acestea nu elimină complet necesitatea implicării umane. Creatorii și editorii de conținut joacă în continuare un rol crucial în definirea strategiei generale de conținut, în furnizarea de îndrumări către LLM și în asigurarea calității și adecvării conținutului generat.

Prin automatizarea aspectelor mai repetitive și consumatoare de timp ale creării de conținut, modelele de Generare Contextuală de Conținut eliberează timp și resurse umane valoroase care pot fi redirecționate către sarcini cu valoare mai mare. Această productivitate crescută vă permite să furnizați conținut mai personalizat și mai angajant utilizatorilor, optimizând în același timp fluxurile de lucru pentru crearea de conținut.

Iterație și Experimentare Rapidă

Modelele de Generare Contextuală de Conținut vă permit să iterați și să experimentați rapid cu diferite variații de conținut, permițând optimizarea și rafinarea mai rapidă a strategiei dvs. de conținut. Puteți genera multiple versiuni de conținut în câteva secunde, simplu prin ajustarea contextului, șabloanelor sau instrucțiunilor furnizate modelului.

Această capacitate de iterație rapidă oferă mai multe beneficii cheie:

1. **Testare și Optimizare**: Cu abilitatea de a genera rapid variații de conținut, puteți testa cu ușurință diferite abordări și măsura eficacitatea acestora. De exemplu, puteți genera multiple versiuni ale unei descrieri de produs sau ale unui mesaj de

marketing, fiecare adaptat unui segment specific de utilizatori sau unui context specific. Prin analizarea metricilor de angajament ale utilizatorilor, precum ratele de click sau ratele de conversie, puteți identifica variațiile de conținut cele mai eficiente și vă puteți optimiza strategia de conținut în consecință.

2. **Testare A/B**: Modelele de Generare Contextuală de Conținut permit testarea A/B fără probleme a conținutului. Puteți genera două sau mai multe variații de conținut și le puteți servi aleatoriu diferitelor grupuri de utilizatori. Prin compararea performanței fiecărei variații, puteți determina care conținut rezonează cel mai bine cu publicul țintă. Această abordare bazată pe date vă permite să luați decizii informate și să vă rafinați continuu conținutul pentru a maximiza angajamentul utilizatorilor și pentru a atinge rezultatele dorite.
3. **Experimente de Personalizare**: Iterația și experimentarea rapidă sunt deosebit de valoroase când vine vorba de personalizare. Cu modelele de Generare Contextuală de Conținut, puteți genera rapid variații de conținut personalizat bazate pe diferite segmente de utilizatori, preferințe sau comportamente. Prin experimentarea cu diferite strategii de personalizare, puteți identifica abordările cele mai eficiente pentru angajarea utilizatorilor individuali și furnizarea de experiențe personalizate.
4. **Adaptarea la Tendințele în Schimbare**: Capacitatea de a itera și experimenta rapid vă permite să rămâneți agili și să vă adaptați la tendințele și preferințele în schimbare ale utilizatorilor. Pe măsură ce apar noi subiecte, cuvinte cheie sau comportamente ale utilizatorilor, puteți genera rapid conținut aliniat acestor tendințe. Prin experimentare și rafinare continuă a conținutului, puteți rămâne relevanți și menține un avantaj competitiv în peisajul digital aflat în continuă evoluție.
5. **Experimentare Rentabilă**: Experimentarea tradițională cu conținutul implică adesea timp și resurse semnificative, deoarece creatorii de conținut trebuie să dezvolte și să testeze manual diferite variații. Cu toate acestea, folosind modelele de Generare Contextuală de Conținut, costul experimentării este mult redus.

> LLM-urile pot genera variații de conținut rapid și la scară, permițându-vă să explorați o gamă largă de idei și abordări fără a suporta costuri substanțiale.

Pentru a profita la maximum de iterația și experimentarea rapidă, este important să aveți un cadru de experimentare bine definit. Acest cadru ar trebui să includă:

- Obiective clare și ipoteze pentru fiecare experiment
- Metrici și mecanisme de urmărire adecvate pentru măsurarea performanței conținutului
- Strategii de segmentare și direcționare pentru a asigura că variațiile relevante de conținut ajung la utilizatorii potriviți
- Instrumente de analiză și raportare pentru a extrage informații din datele experimentale
- Un proces pentru încorporarea învățămintelor și optimizărilor în strategia de conținut

Prin adoptarea iterației și experimentării rapide, puteți rafina și optimiza continuu conținutul, asigurându-vă că rămâne captivant, relevant și eficient în atingerea obiectivelor aplicației dvs. Această abordare agilă a creării de conținut vă permite să rămâneți în fața competiției și să oferiți experiențe excepționale utilizatorilor.

Scalabilitate și Eficiență

Pe măsură ce aplicațiile cresc și cererea pentru conținut personalizat crește, modelele de generare contextuală de conținut permit scalarea eficientă a creării de conținut. LLM-urile pot genera conținut pentru un număr mare de utilizatori și contexte simultan, fără a necesita o creștere proporțională a resurselor umane. Această scalabilitate permite aplicațiilor să ofere experiențe personalizate unei baze de utilizatori în creștere fără a suprasolicita capacitățile lor de creare de conținut.

Rețineți că generarea contextuală de conținut poate fi utilizată eficient pentru a internaționaliza aplicația dvs. "din mers". De fapt, exact asta am făcut eu folosind Gemul meu Instant18n pentru a livra Olympia în mai mult de o jumătate de duzină de limbi, deși avem mai puțin de un an.

Localizare Bazată pe AI

Dacă îmi permiteți să mă laud pentru un moment, cred că biblioteca mea Instant18n pentru aplicații Rails este un exemplu revoluționar al modelului "Generare Contextuală de Conținut" în acțiune, demonstrând potențialul transformator al AI în dezvoltarea aplicațiilor. Acest gem folosește puterea modelului lingvistic GPT de la OpenAI pentru a revoluționa modul în care sunt gestionate internaționalizarea și localizarea în aplicațiile Rails.

În mod tradițional, internaționalizarea unei aplicații Rails implică definirea manuală a cheilor de traducere și furnizarea traducerilor corespunzătoare pentru fiecare limbă suportată. Acest proces poate fi consumator de timp, intensiv în resurse și predispus la inconsistențe. Cu toate acestea, cu gemul Instant18n, paradigma localizării este complet redefinită.

Prin integrarea unui model lingvistic de mari dimensiuni, gemul Instant18n vă permite să generați traduceri din mers, bazate pe contextul și semnificația textului. În loc să se bazeze pe chei de traducere predefinite și traduceri statice, gemul traduce dinamic textul folosind puterea AI. Această abordare oferă mai multe beneficii cheie:

1. **Localizare Fără Efort**: Cu gemul Instant18n, dezvoltatorii nu mai trebuie să definească și să întrețină manual fișiere de traducere pentru fiecare limbă suportată. Gemul generează automat traduceri bazate pe textul furnizat și limba țintă dorită, făcând procesul de localizare fără efort și fluid.

2. **Acuratețe Contextuală**: AI-ului i se poate oferi suficient context pentru a înțelege nuanțele textului tradus. Poate lua în considerare contextul înconjurător, expresiile idiomatice și referințele culturale pentru a genera traduceri care sunt precise, naturale și contextual adecvate.
3. **Suport Extins pentru Limbi**: Gemul Instant18n folosește vastele cunoștințe și capacități lingvistice ale GPT, permițând traduceri într-o gamă extinsă de limbi. De la limbi comune precum spaniola și franceza până la limbi mai obscure sau fictive precum klingoniana și elfica, gemul poate gestiona o varietate largă de cerințe de traducere.
4. **Flexibilitate și Creativitate**: Gemul merge dincolo de traducerile lingvistice tradiționale și permite opțiuni de localizare creative și neconvenționale. Dezvoltatorii pot traduce text în diverse stiluri, dialecte sau chiar limbi fictive, deschizând noi posibilități pentru experiențe unice ale utilizatorilor și conținut captivant.
5. **Optimizarea Performanței**: Gemul Instant18n încorporează mecanisme de memorare în cache pentru a îmbunătăți performanța și a reduce costurile traducerilor repetate. Textul tradus este memorat în cache, permițând ca cererile ulterioare pentru aceeași traducere să fie servite rapid fără necesitatea unor apeluri API redundante.

Gemul Instant18n exemplifică puterea modelului "Generare Contextuală de Conținut" prin utilizarea AI pentru a genera conținut localizat în mod dinamic. Acesta demonstrează cum AI poate fi integrat în funcționalitatea de bază a unei aplicații Rails, transformând modul în care dezvoltatorii abordează internaționalizarea și localizarea.

Prin eliminarea necesității gestionării manuale a traducerilor și permiterea traducerilor instantanee bazate pe context, gem-ul Instant18n economisește timp și efort semnificativ pentru dezvoltatori. Le permite acestora să se concentreze pe construirea funcționalităților principale ale aplicației lor, asigurând în același timp că aspectul de localizare este gestionat fluid și precis.

Importanţa Testării cu Utilizatori şi a Feedback-ului

În cele din urmă, țineți întotdeauna cont de importanța testării cu utilizatori și a feedback-ului. Este crucial să validați că generarea de conținut contextual îndeplinește așteptările utilizatorilor și se aliniază cu obiectivele aplicației. Iterați și rafinați continuu conținutul generat pe baza informațiilor de la utilizatori și a datelor analitice. Dacă generați conținut dinamic la scară largă care ar fi imposibil de validat manual de către dumneavoastră și echipa dumneavoastră, luați în considerare adăugarea unor mecanisme de feedback care să permită utilizatorilor să raporteze conținutul care este ciudat sau greșit, împreună cu o explicație a motivului. Acest feedback prețios poate fi chiar transmis unui sistem AI însărcinat cu efectuarea ajustărilor la componenta care a generat conținutul!

Interface Generativă

Atenţia este atât de preţioasă în zilele noastre încât implicarea eficientă a utilizatorilor necesită acum experienţe software care nu sunt doar fluide şi intuitive, ci şi foarte personalizate pentru nevoile, preferinţele şi contextele individuale. Drept urmare, designerii şi dezvoltatorii se confruntă tot mai mult cu provocarea de a crea interfeţe pentru utilizatori care se pot adapta şi răspunde cerinţelor unice ale fiecărui utilizator *la scară largă*.

Interfaţa Generativă (GenUI) este o abordare cu adevărat revoluţionară a designului interfeţei utilizator care valorifică puterea modelelor lingvistice mari (LLM) pentru a crea experienţe de utilizare foarte personalizate şi dinamice din mers. Am vrut să mă asigur că vă ofer cel puţin o introducere în GenUI în această carte, deoarece cred că este una dintre oportunităţile cele mai promiţătoare care există în prezent în domeniul designului şi framework-urilor pentru aplicaţii. Sunt convins că zeci sau mai multe proiecte comerciale şi open-source de succes vor apărea în această nişă particulară.

În esență, GenUI combină principiile Generării de Conținut Contextual cu tehnici avansate de AI pentru a genera dinamic elemente de interfață utilizator, cum ar fi text, imagini și layouturi, bazate pe o înțelegere profundă a contextului, preferințelor și obiectivelor utilizatorului. GenUI permite designerilor și dezvoltatorilor să creeze interfețe care se adaptează și evoluează ca răspuns la interacțiunile utilizatorului, oferind un nivel de personalizare care anterior era imposibil de atins.

GenUI reprezintă o schimbare fundamentală în modul în care abordăm designul interfeței utilizator. În loc să proiectăm pentru mase, GenUI ne permite să proiectăm pentru individ. Conținutul și interfețele personalizate au potențialul de a crea experiențe de utilizare care rezonează cu fiecare utilizator la un nivel mai profund, crescând implicarea, satisfacția și loialitatea.

Ca tehnică de ultimă generație, tranziția către GenUI este plină de provocări conceptuale și practice. Integrarea AI-ului în procesul de design, asigurarea că interfețele generate sunt nu doar personalizate, ci și utilizabile, accesibile și aliniate cu brandul și experiența generală a utilizatorului, toate acestea sunt provocări care fac din GenUI o preocupare pentru puțini, nu pentru mulți. În plus, implicarea AI-ului ridică întrebări despre confidențialitatea datelor, transparență și chiar implicații etice.

În ciuda provocărilor, experiențele personalizate la scară au puterea de a transforma complet modul în care interacționăm cu produsele și serviciile digitale. Deschide posibilități pentru crearea unor interfețe incluzive și accesibile care răspund nevoilor diverse ale utilizatorilor, indiferent de abilitățile, background-ul sau preferințele lor.

În acest capitol, vom explora conceptul de GenUI, examinând câteva caracteristici definitorii, beneficii cheie și potențiale provocări. Începem prin a analiza cea mai basic și accesibilă formă de GenUI: generarea textului pentru interfețele utilizator proiectate și implementate în mod tradițional.

Generarea Textului pentru Interfeţele Utilizator

Elementele de text care există în interfața vizuală a aplicației dvs., cum ar fi etichetele formularelor, indiciile explicative și textul explicativ, sunt de obicei codate fix în șabloane sau componente UI, oferind o experiență consistentă dar generică pentru toți utilizatorii. Folosind pattern-uri de generare de conținut contextual, puteți transforma aceste elemente statice în componente dinamice, sensibile la context și personalizate.

Formulare Personalizate

Formularele sunt o parte omniprezentă a aplicațiilor web și mobile, servind ca principal mijloc de colectare a datelor de la utilizatori. Cu toate acestea, formularele tradiționale oferă adesea o experiență generică și impersonală, cu etichete și câmpuri standard care nu se aliniază întotdeauna cu contextul sau nevoile specifice ale utilizatorului. Utilizatorii sunt mai predispuși să completeze formulare care par adaptate nevoilor și preferințelor lor, ducând la rate de conversie și satisfacție mai mari.

Cu toate acestea, este important să găsim un echilibru între personalizare și consistență. În timp ce adaptarea formularelor la utilizatori individuali poate fi benefică, este crucial să menținem un nivel de familiaritate și predictibilitate. Utilizatorii ar trebui să poată în continuare să recunoască și să navigheze formularele cu ușurință, chiar și cu elemente personalizate.

Iată câteva idei de formulare personalizate pentru inspirație:

Sugestii Contextuale pentru Câmpuri

GenUI poate analiza interacțiunile anterioare ale utilizatorului, preferințele și datele pentru a oferi sugestii inteligente pentru câmpuri ca predicții. De exemplu, dacă utilizatorul și-a introdus anterior adresa de livrare, formularul poate completa automat câmpurile relevante cu informațiile salvate. Acest lucru nu doar economisește timp, dar demonstrează și că aplicația înțelege și își amintește preferințele utilizatorului.

Stai puțin, nu este această tehnică ceva ce ar putea fi făcut și fără implicarea AI-ului? Desigur, dar frumusețea implementării acestui tip de funcționalitate cu AI constă în două aspecte: 1) cât de ușor poate fi implementată și 2) cât de rezistentă poate fi pe măsură ce UI-ul tău se schimbă și evoluează în timp.

Hai să creăm rapid un serviciu pentru sistemul nostru teoretic de gestionare a comenzilor, care încearcă să completeze în mod proactiv adresa corectă de livrare pentru utilizator.

```
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
```

Acest exemplu este foarte simplificat, dar ar trebui să funcționeze în majoritatea cazurilor. Ideea este să lăsăm inteligența artificială să facă o presupunere în același mod în care ar face-o un om. Pentru a clarifica despre ce vorbesc, să analizăm niște date exemplu:

```
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"}
]
```

Ai observat tiparul din date? Îți garantez că acestea sunt lucruri ușoare pentru un LLM. Pentru a demonstra acest lucru, să întrebăm GPT-4 care este cea mai probabilă adresă de livrare pentru un "termometru".

```
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.
```

Dacă te gândești că ar fi exagerat să folosești un model costisitor precum GPT-4 pentru această sarcină, ai dreptate! Am încercat același prompt pe Mistral 7B Instruct și a produs următorul răspuns cu o viteză de 75 de tokenuri pe secundă și un cost insignifiant de $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.
```

Merită efortul și costul acestei tehnici pentru a face experiența de checkout mai magică? Pentru mulți comercianți online, absolut. Și din ce se poate observa, costul calculului AI va continua să scadă, mai ales pentru furnizorii de găzduire a modelelor open source aflați într-o cursă spre cele mai mici prețuri.

Folosește un Șablon de Prompt și StructuredIO împreună cu Response Fencing pentru a optimiza acest tip de completare prin chat.

Ordonarea Adaptivă a Câmpurilor

Ordinea în care sunt prezentate câmpurile formularului poate avea un impact semnificativ asupra experienței utilizatorului și a ratelor de completare. Cu GenUI, poți ajusta dinamic ordinea câmpurilor în funcție de contextul utilizatorului și importanța fiecărui câmp. De exemplu, dacă utilizatorul completează un formular de înregistrare pentru o aplicație de fitness, formularul ar putea prioritiza câmpurile legate de obiectivele și preferințele lor de fitness, făcând procesul mai relevant și mai angajant.

Microcopy Personalizat

Textul instructiv, mesajele de eroare și alte elemente de microcopy asociate formularelor pot fi, de asemenea, personalizate folosind GenUI. În loc să afișeze mesaje de eroare generice precum "Adresă de email invalidă", poți genera mesaje mai utile și contextuale precum "Te rugăm să introduci o adresă de email validă pentru a primi confirmarea comenzii tale". Aceste elemente personalizate pot face experiența formularului mai prietenoasă și mai puțin frustrantă.

Validare Personalizată

În aceeași direcție cu Microcopy-ul Personalizat, ai putea folosi AI pentru a valida formularul în moduri care par magice. Imaginează-ți că permiți unui AI să valideze un formular de profil utilizator, căutând potențiale greșeli la nivel *semantic*.

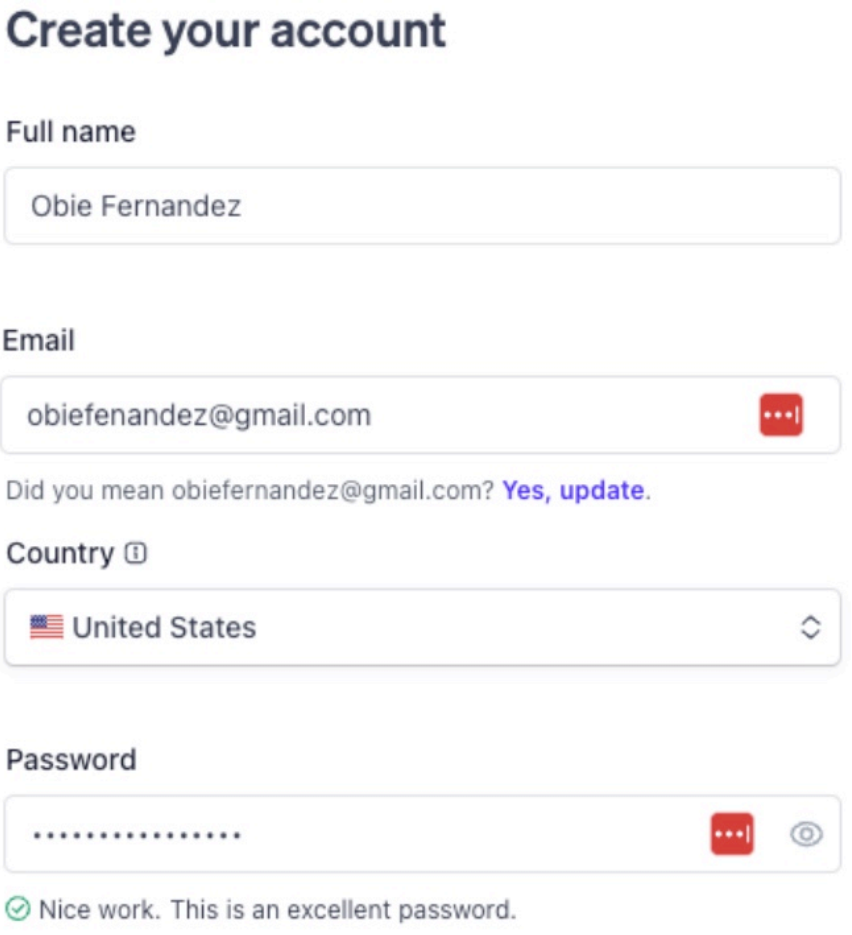


Figura 8. Poți observa validarea semantică în acțiune?

Dezvăluire Progresivă

GenUI poate determina în mod inteligent care câmpuri ale formularului sunt esențiale în funcție de contextul utilizatorului și poate dezvălui treptat câmpuri suplimentare după necesitate. Această tehnică de dezvăluire progresivă ajută la reducerea încărcării cognitive și face procesul de completare a formularului mai ușor de gestionat. De

exemplu, dacă un utilizator se înscrie pentru un abonament de bază, formularul poate prezenta inițial doar câmpurile esențiale, iar pe măsură ce utilizatorul progresează sau selectează anumite opțiuni, câmpuri suplimentare relevante pot fi introduse dinamic.

Text Explicativ Sensibil la Context

Tooltip-urile sunt adesea utilizate pentru a oferi informații suplimentare sau îndrumări utilizatorilor când aceștia trec cu mouse-ul peste sau interacționează cu anumite elemente. Cu o abordare de "Generare de Conținut Contextual", poți genera tooltip-uri care se adaptează la contextul utilizatorului și oferă informații relevante. De exemplu, dacă un utilizator explorează o funcționalitate complexă, tooltip-ul poate oferi sfaturi personalizate sau exemple bazate pe interacțiunile lor anterioare sau nivelul de experiență.

Textul explicativ, cum ar fi instrucțiunile, descrierile sau mesajele de ajutor, poate fi generat dinamic în funcție de contextul utilizatorului. În loc să prezinte explicații generice, poți utiliza LLM-uri pentru a genera text adaptat nevoilor sau întrebărilor specifice ale utilizatorului. De exemplu, dacă un utilizator întâmpină dificultăți cu un anumit pas într-un proces, textul explicativ poate oferi îndrumări personalizate sau sfaturi de depanare.

Microcopy se referă la fragmentele mici de text care ghidează utilizatorii prin aplicația ta, cum ar fi etichetele butoanelor, mesajele de eroare sau prompturile de confirmare. Prin aplicarea abordării de Generare de Conținut Contextual la microcopy, poți crea o interfață utilizator adaptivă care răspunde la acțiunile utilizatorului și oferă text relevant și util. De exemplu, dacă un utilizator este pe cale să efectueze o acțiune critică, promptul de confirmare poate fi generat dinamic pentru a oferi un mesaj clar și personalizat.

Textul explicativ personalizat și tooltip-urile pot îmbunătăți semnificativ procesul de integrare pentru utilizatorii noi. Prin furnizarea de îndrumări și exemple specifice contextului, poți ajuta utilizatorii să înțeleagă și să navigheze rapid prin aplicație, reducând curba de învățare și crescând adopția.

Elementele chrome dinamice și sensibile la context pot face, de asemenea, ca aplicația să pară mai intuitivă și mai atractivă. Utilizatorii sunt mai predispuși să interacționeze cu și să exploreze funcționalitățile când textul însoțitor este adaptat nevoilor și intereselor lor specifice.

Până acum am discutat despre idei pentru îmbunătățirea paradigmelor UI existente cu ajutorul AI, dar ce-ar fi să regândim modul în care sunt proiectate și implementate interfețele utilizator într-un mod mai radical?

Definirea UI-ului Generativ

Spre deosebire de designul UI tradițional, unde designerii creează interfețe fixe și statice, GenUI sugerează un viitor în care software-ul nostru se poate lăuda cu experiențe flexibile și personalizate care pot evolua și se pot adapta în timp real. De fiecare dată când folosim o interfață conversațională bazată pe AI, permitem AI-ului să se adapteze la nevoile particulare ale utilizatorului. GenUI face un pas mai departe prin aplicarea acestui nivel de adaptabilitate la interfața *vizuală* a software-ului.

Motivul pentru care este posibil să experimentăm cu ideile GenUI astăzi este că modelele lingvistice de mari dimensiuni înțeleg deja programarea și cunoștințele lor de bază includ tehnologii și framework-uri UI. Întrebarea este dacă modelele lingvistice de mari dimensiuni pot fi folosite pentru a genera elemente UI, cum ar fi text, imagini, layout-uri și chiar interfețe complete, care sunt adaptate fiecărui utilizator în parte. Modelul ar putea fi instruit să ia în considerare diverși factori, cum ar fi interacțiunile anterioare ale utilizatorului, preferințele declarate, informațiile demografice și contextul actual de utilizare, pentru a crea interfețe extrem de personalizate și relevante.

GenUI diferă de designul tradițional al interfeței utilizator în mai multe aspecte cheie:

1. **Dinamic și Adaptabil**: Designul UI tradițional implică crearea unor interfețe fixe, statice, care rămân aceleași pentru toți utilizatorii. În schimb, GenUI permite interfețe care se pot adapta și schimba dinamic în funcție de nevoile și contextul utilizatorului. Acest lucru înseamnă că aceeași aplicație poate prezenta interfețe diferite pentru utilizatori diferiți sau chiar pentru același utilizator în situații diferite.
2. **Personalizare la Scară**: Cu designul tradițional, crearea de experiențe personalizate pentru fiecare utilizator este adesea impracticabilă din cauza timpului și resurselor necesare. Pe de altă parte, GenUI permite personalizarea la scară largă. Prin utilizarea AI-ului, designerii pot crea interfețe care se adaptează automat la nevoile și preferințele unice ale fiecărui utilizator, fără a fi nevoie să proiecteze și să dezvolte manual interfețe separate pentru fiecare segment de utilizatori.
3. **Focalizare pe Rezultate**: Designul UI tradițional se concentrează adesea pe crearea unor interfețe vizual atractive și funcționale. Deși aceste aspecte rămân importante în GenUI, focusul principal se mută către obținerea rezultatelor dorite de utilizator. GenUI își propune să creeze interfețe care sunt optimizate pentru obiectivele și sarcinile specifice ale fiecărui utilizator, prioritizând utilizabilitatea și eficacitatea în detrimentul considerentelor pur estetice.
4. **Învățare și Îmbunătățire Continuă**: Sistemele GenUI pot învăța și se pot îmbunătăți continuu în timp, bazându-se pe interacțiunile și feedback-ul utilizatorilor. Pe măsură ce utilizatorii interacționează cu interfețele generate, modelele AI pot colecta date despre comportamentul, preferințele și rezultatele utilizatorilor, folosind aceste informații pentru a rafina și optimiza generările viitoare de interfețe. Acest proces iterativ de învățare permite sistemelor GenUI să devină din ce în ce mai eficiente în satisfacerea nevoilor utilizatorilor în timp.

Este important de menționat că GenUI nu este același lucru cu instrumentele de design asistate de AI, cum ar fi cele care oferă sugestii sau automatizează anumite sarcini de design. În timp ce aceste instrumente pot fi utile în eficientizarea procesului de design,

ele încă se bazează pe designeri pentru a lua decizii finale și a crea interfețe statice. GenUI, pe de altă parte, implică ca sistemul AI să aibă un rol mai activ în generarea și adaptarea efectivă a interfețelor pe baza datelor și contextului utilizatorului.

GenUI reprezintă o schimbare semnificativă în modul în care abordăm designul interfeței utilizator, îndepărtându-ne de soluțiile universale și îndreptându-ne către experiențe foarte personalizate și adaptabile. Prin valorificarea puterii AI, GenUI are potențialul de a revoluționa modul în care interacționăm cu produsele și serviciile digitale, creând interfețe care sunt mai intuitive, mai angajante și mai eficiente pentru fiecare utilizator în parte.

Exemplu

Pentru a ilustra conceptul de GenUI, să luăm în considerare o aplicație ipotetică de fitness numită "FitAI". Această aplicație își propune să ofere planuri de antrenament personalizate și sfaturi nutriționale utilizatorilor pe baza obiectivelor, nivelurilor de fitness și preferințelor individuale.

Într-o abordare tradițională de design UI, FitAI ar putea avea un set fix de ecrane și elemente care sunt aceleași pentru toți utilizatorii. Cu toate acestea, cu GenUI, interfața aplicației s-ar putea adapta dinamic la nevoile și contextul unic al fiecărui utilizator.

Această abordare este destul de greu de imaginat că ar putea fi implementată în 2024 și s-ar putea să nu aibă nici măcar un ROI adecvat, dar este posibilă.

Iată cum ar putea funcționa:

1. **Onboarding**:

    - În loc de un chestionar standard, FitAI folosește un AI conversațional pentru a colecta informații despre obiectivele utilizatorului, nivelul actual de fitness și preferințe.

- Pe baza acestei interacțiuni inițiale, AI-ul generează un layout personalizat pentru dashboard, evidențiind funcționalitățile și informațiile cele mai relevante pentru obiectivele utilizatorului.
- Tehnologia AI actuală ar putea avea la dispoziție o selecție de componente de ecran pentru a le utiliza în compunerea dashboard-ului personalizat.
- Tehnologia AI viitoare ar putea prelua rolul unui designer UI experimentat și să creeze efectiv dashboard-ul *de la zero*.

2. **Planificator de antrenamente**:

- Interfața planificatorului de antrenamente este adaptată de AI pentru a se potrivi specific nivelului de experiență al utilizatorului și echipamentului disponibil.
- Pentru un începător fără echipament, ar putea arăta exerciții simple cu greutatea corpului, însoțite de instrucțiuni detaliate și videoclipuri.
- Pentru un utilizator avansat cu acces la o sală de fitness, ar putea afișa rutine mai complexe cu mai puțin conținut explicativ.
- Conținutul planificatorului de antrenamente nu este doar filtrat dintr-un set mai mare. Acesta poate fi generat *din mers* pe baza unei baze de cunoștințe care este interogată cu un context ce include tot ce se știe despre utilizator.

3. **Monitorizarea progresului**:

- Interfața de monitorizare a progresului evoluează în funcție de obiectivele utilizatorului și tiparele de implicare.
- Dacă un utilizator se concentrează în principal pe pierderea în greutate, interfața ar putea afișa prominent un grafic al tendinței greutății și statistici privind arderea caloriilor.
- Pentru un utilizator care își construiește masa musculară, ar putea evidenția creșterile de forță și modificările compoziției corporale.

- AI-ul poate adapta această parte a aplicației la progresul real al utilizatorului. Dacă progresul se oprește pentru o perioadă de timp, aplicația poate trece într-un mod în care încearcă să determine utilizatorul să dezvăluie motivele acestui regres, pentru a le atenua.

4. **Sfaturi nutriționale**:

   - Secțiunea de nutriție se adaptează la preferințele și restricțiile alimentare ale utilizatorului.
   - Pentru un utilizator vegan, ar putea arăta sugestii de mese pe bază de plante și surse de proteine.
   - Pentru un utilizator cu intoleranță la gluten, ar filtra automat alimentele care conțin gluten din recomandări.
   - Din nou, conținutul nu este extras dintr-un set masiv de date despre mese care se aplică tuturor utilizatorilor, ci este sintetizat dintr-o bază de cunoștințe care conține informații adaptabile bazate pe situația și constrângerile specifice ale utilizatorului.
   - De exemplu, rețetele sunt generate cu specificații ale ingredientelor care se potrivesc nevoilor calorice în continuă schimbare ale utilizatorului, pe măsură ce nivelul său de fitness și statisticile corporale evoluează.

5. **Elemente motivaționale**:

   - Conținutul motivațional și notificările aplicației sunt personalizate în funcție de tipul de personalitate al utilizatorului și răspunsul la diferite strategii motivaționale.
   - Unii utilizatori ar putea primi mesaje încurajatoare, în timp ce alții primesc feedback mai orientat spre date.

În acest exemplu, GenUI permite FitAI să creeze o experiență foarte personalizată pentru fiecare utilizator, potențial crescând implicarea, satisfacția și probabilitatea de a atinge obiectivele de fitness. Elementele interfeței, conținutul și chiar "personalitatea"

aplicației se adaptează pentru a servi cel mai bine nevoilor și preferințelor fiecărui utilizator în parte.

Trecerea la designul orientat spre rezultate

GenUI reprezintă o schimbare fundamentală în abordarea designului interfeței utilizator!, trecând de la focusarea pe crearea elementelor specifice de interfață la o abordare mai holistică, orientată spre rezultate. Această schimbare are mai multe implicații importante:

1. **Concentrarea pe obiectivele utilizatorului**:

    - Designerii vor trebui să se gândească mai profund la obiectivele utilizatorilor și rezultatele dorite, mai degrabă decât la componente specifice ale interfeței.
    - Accentul va fi pus pe crearea unor sisteme care pot genera interfețe care ajută utilizatorii să-și atingă obiectivele în mod eficient și eficace.
    - Vor apărea noi cadre de lucru UI care vor oferi designerilor bazați pe AI instrumentele necesare pentru a putea genera experiențe de utilizare *din mers* și *de la zero* în loc să se bazeze pe specificații predefinite ale ecranelor.

2. **Schimbarea rolului designerilor**:

    - Designerii vor trece de la crearea de layout-uri fixe la definirea regulilor, constrângerilor și ghidurilor pe care sistemele AI să le urmeze când generează interfețe.
    - Vor trebui să dezvolte abilități în domenii precum analiza datelor, ingineria prompturilor AI și gândirea sistemică pentru a ghida eficient sistemele GenUI.

3. **Importanța cercetării utilizatorilor**:

- Cercetarea utilizatorilor devine și mai critică într-un context GenUI, deoarece designerii trebuie să înțeleagă nu doar preferințele utilizatorilor, ci și modul în care aceste preferințe și nevoi se schimbă în diferite contexte.
- Testarea continuă cu utilizatorii și buclele de feedback vor fi esențiale pentru a rafina și îmbunătăți capacitatea AI-ului de a genera interfețe eficiente.

4. **Proiectarea pentru variabilitate**:

   - În loc să creeze o singură interfață "perfectă", designerii vor trebui să ia în considerare multiple variații posibile și să se asigure că sistemul poate genera interfețe adecvate pentru diverse nevoi ale utilizatorilor.
   - Aceasta include proiectarea pentru cazuri limită și asigurarea că interfețele generate mențin utilizabilitatea și accesibilitatea în diferite configurații.
   - Diferențierea produselor capătă noi dimensiuni implicând perspective divergente asupra psihologiei utilizatorului și valorificarea unor seturi de date și baze de cunoștințe unice indisponibile concurenților.

Provocări şi consideraţii

În timp ce GenUI oferă posibilități interesante, prezintă și mai multe provocări și considerații:

1. **Limitări tehnice**:

   - Tehnologia AI actuală, deși avansată, încă are limitări în înțelegerea intențiilor complexe ale utilizatorilor și generarea de interfețe cu adevărat conștiente de context.
   - Probleme de performanță legate de generarea în timp real a elementelor de interfață, în special pe dispozitive mai puțin puternice.

2. **Cerințe privind datele**:

- În funcție de cazul de utilizare, sistemele GenUI eficiente ar putea necesita cantități semnificative de date despre utilizatori pentru a genera interfețe personalizate.
- Provocările legate de obținerea etică a datelor autentice ale utilizatorilor ridică îngrijorări cu privire la confidențialitatea și securitatea datelor, precum și potențialele prejudecăți în datele folosite pentru antrenarea modelelor GenUI.

3. **Utilizabilitate și consecvență**:

   - Cel puțin până când practica devine larg răspândită, o aplicație cu interfețe în continuă schimbare ar putea duce la probleme de utilizare, deoarece utilizatorii ar putea avea dificultăți în găsirea elementelor familiare sau în navigarea eficientă.
   - Va fi crucial să se găsească un echilibru între personalizare și menținerea unei interfețe consecvente și ușor de învățat.

4. **Dependența excesivă de AI**:

   - Există riscul de a delega excesiv deciziile de design sistemelor AI, ceea ce poate duce la alegeri de interfață neinspirante, problematice sau pur și simplu defecte.
   - Supravegherea umană și capacitatea de a suprascrie designurile generate de AI vor rămâne importante în viitorul apropiat.

5. **Probleme de accesibilitate**:

   - Asigurarea că interfețele generate dinamic rămân accesibile utilizatorilor cu dizabilități prezintă provocări complet noi, ceea ce este îngrijorător având în vedere nivelul scăzut de conformitate cu standardele de accesibilitate demonstrat de sistemele tipice.

- Pe de altă parte, designerii AI pot fi implementați cu preocupare *încorporată* pentru accesibilitate și cu capabilități de construire a interfețelor accesibile din mers, la fel cum construiesc interfețe pentru utilizatorii fără dizabilități.
- În orice caz, sistemele GenUI ar trebui să fie proiectate cu ghiduri și procese robuste de testare a accesibilității.

6. **Încrederea și transparența utilizatorilor**:

    - Utilizatorii s-ar putea simți inconfortabil cu interfețele care par să "știe prea multe" despre ei sau care se schimbă în moduri pe care nu le înțeleg.
    - Va fi important să se asigure transparența cu privire la modul și motivul pentru care interfețele sunt personalizate pentru a construi încrederea utilizatorilor.

Perspective de viitor şi oportunităţi

Viitorul Interfeței Utilizator Generative (GenUI) promite să revoluționeze modul în care interacționăm cu produsele și serviciile digitale. Pe măsură ce această tehnologie continuă să evolueze, putem anticipa o schimbare seismică în modul în care interfețele utilizator sunt proiectate, implementate și experimentate. Cred că GenUI este fenomenul care va împinge în sfârșit software-ul nostru în tărâmul a ceea ce este considerat acum science fiction.

Una dintre cele mai interesante perspective ale GenUI este potențialul său de a îmbunătăți accesibilitatea la o scară grandioasă care depășește simpla asigurare că persoanele cu dizabilități grave nu sunt complet excluse de la utilizarea software-ului. Prin adaptarea automată a interfețelor la nevoile individuale ale utilizatorilor, GenUI ar putea face experiențele digitale mai incluzive ca niciodată. Imaginați-vă interfețe care se ajustează fără probleme pentru a oferi text mai mare pentru utilizatorii tineri sau cu deficiențe de vedere sau layouturi simplificate pentru cei cu dizabilități cognitive, toate fără a necesita configurare manuală sau versiuni separate "accesibile" ale aplicațiilor.

Capacitățile de personalizare ale GenUI vor conduce probabil la creșterea implicării, satisfacției și loialității utilizatorilor în cazul unei game largi de produse digitale. Pe măsură ce interfețele devin mai adaptate la preferințele și comportamentele individuale, utilizatorii vor găsi experiențele digitale mai intuitive și mai plăcute, conducând potențial la interacțiuni mai profunde și mai semnificative cu tehnologia.

GenUI are, de asemenea, potențialul de a transforma procesul de integrare pentru noii utilizatori. Prin crearea unor experiențe intuitive și personalizate pentru utilizatorii care folosesc aplicația pentru prima dată și care se adaptează rapid la nivelul de expertiză al fiecărui utilizator, GenUI ar putea reduce semnificativ curba de învățare asociată cu aplicațiile noi. Acest lucru ar putea duce la rate mai rapide de adoptare și la creșterea încrederii utilizatorilor în explorarea noilor caracteristici și funcționalități.

O altă posibilitate interesantă este capacitatea GenUI de a menține o experiență de utilizare consecventă pe diferite dispozitive și platforme, optimizând în același timp pentru fiecare context specific de utilizare. Acest lucru ar putea rezolva provocarea de lungă durată de a oferi experiențe coerente într-un peisaj din ce în ce mai fragmentat al dispozitivelor, de la smartphone-uri și tablete la computere desktop și tehnologii emergente precum ochelarii de realitate augmentată.

Natura bazată pe date a GenUI deschide oportunități pentru iterație și îmbunătățire rapidă în designul UI. Prin colectarea de date în timp real despre modul în care utilizatorii interacționează cu interfețele generate, designerii și dezvoltatorii pot obține perspective fără precedent asupra comportamentului și preferințelor utilizatorilor. Această buclă de feedback ar putea duce la îmbunătățiri continue în designul UI, bazate pe modele reale de utilizare mai degrabă decât pe presupuneri sau testare limitată cu utilizatori.

Pentru a se pregăti pentru această schimbare, designerii vor trebui să își dezvolte seturile de abilități și mentalitățile. Accentul se va muta de la crearea de layouturi fixe la dezvoltarea unor sisteme și ghiduri de design cuprinzătoare care pot informa generarea de interfețe bazată pe AI. Designerii vor trebui să cultive o înțelegere profundă a analizei

datelor, tehnologiilor AI şi gândirii sistemice pentru a ghida eficient sistemele GenUI.

Mai mult, pe măsură ce GenUI estompează graniţele dintre design şi tehnologie, designerii vor trebui să colaboreze mai strâns cu dezvoltatorii şi specialiştii în ştiinţa datelor. Această abordare interdisciplinară va fi crucială în crearea unor sisteme GenUI care sunt nu doar atractive vizual şi uşor de utilizat, ci şi robuste din punct de vedere tehnic şi etice.

Implicaţiile etice ale GenUI vor ajunge, de asemenea, în prim-plan pe măsură ce tehnologia se maturizează. Designerii vor juca un rol crucial în dezvoltarea cadrelor de lucru pentru utilizarea responsabilă a inteligenţei artificiale în designul de interfaţă, asigurându-se că personalizarea îmbunătăţeşte experienţele utilizatorilor fără a compromite confidenţialitatea sau a manipula comportamentul utilizatorului în moduri lipsite de etică.

Privind spre viitor, GenUI prezintă atât oportunităţi incitante, cât şi provocări semnificative. Are potenţialul de a crea experienţe digitale mai intuitive, mai eficiente şi mai satisfăcătoare pentru utilizatori din întreaga lume. Deşi va necesita ca designerii să se adapteze şi să dobândească noi competenţe, oferă totodată o oportunitate fără precedent de a modela viitorul interacţiunii om-calculator în moduri profunde şi semnificative. Călătoria către sisteme GenUI pe deplin realizate va fi fără îndoială complexă, dar recompensele potenţiale în ceea ce priveşte îmbunătăţirea experienţelor utilizatorilor şi accesibilitatea digitală fac ca acest viitor să merite efortul depus.

Orchestrarea Inteligentă a Fluxurilor de Lucru

În domeniul dezvoltării de aplicații, fluxurile de lucru joacă un rol crucial în definirea modului în care sarcinile, procesele și interacțiunile utilizatorilor sunt structurate și executate. Pe măsură ce aplicațiile devin mai complexe și așteptările utilizatorilor continuă să crească, necesitatea unei orchestrări inteligente și adaptive a fluxurilor de lucru devine tot mai evidentă.

Abordarea "Orchestrării Inteligente a Fluxurilor de Lucru" se concentrează pe utilizarea componentelor de IA pentru a orchestra și optimiza dinamic fluxurile de lucru complexe din cadrul aplicațiilor. Scopul este de a crea aplicații mai eficiente, mai reactive și mai adaptabile la datele și contextul în timp real.

În acest capitol, vom explora principiile și tiparele cheie care stau la baza abordării orchestrării inteligente a fluxurilor de lucru. Vom analiza modul în care IA poate

fi utilizată pentru a dirija inteligent sarcinile, pentru a automatiza luarea deciziilor și pentru a adapta dinamic fluxurile de lucru în funcție de diverși factori, cum ar fi comportamentul utilizatorilor, performanța sistemului și regulile de business. Prin exemple practice și scenarii din lumea reală, vom demonstra potențialul transformator al IA în eficientizarea și optimizarea fluxurilor de lucru ale aplicațiilor.

Indiferent dacă construiți aplicații enterprise cu procese de business complexe sau aplicații destinate consumatorilor cu parcursuri dinamice ale utilizatorilor, tiparele și tehnicile discutate în acest capitol vă vor oferi cunoștințele și instrumentele necesare pentru a crea fluxuri de lucru inteligente și eficiente care îmbunătățesc experiența generală a utilizatorului și generează valoare pentru business.

Necesitatea în Business

Abordările tradiționale ale gestionării fluxurilor de lucru se bazează adesea pe reguli predefinite și arbori decizionali statici, care pot fi rigizi, inflexibili și incapabili să facă față naturii dinamice a aplicațiilor moderne.

Să luăm în considerare un scenariu în care o aplicație de comerț electronic trebuie să gestioneze un proces complex de procesare a comenzilor. Fluxul de lucru poate implica mai mulți pași, cum ar fi validarea comenzii, verificarea stocului, procesarea plății, expedierea și notificările către clienți. Fiecare pas poate avea propriul set de reguli, dependențe, integrări externe și mecanisme de gestionare a excepțiilor. Gestionarea unui astfel de flux de lucru manual sau prin logică hardcodată poate deveni rapid complicată, predispusă la erori și dificil de întreținut.

Mai mult, pe măsură ce aplicația se scalează și numărul de utilizatori concurenți crește, fluxul de lucru poate avea nevoie să se adapteze și să se optimizeze în funcție de datele în timp real și performanța sistemului. De exemplu, în perioadele de trafic intens, aplicația poate avea nevoie să ajusteze dinamic fluxul de lucru pentru a prioritiza anumite sarcini, a aloca eficient resursele și a asigura o experiență fluidă utilizatorului.

Aici intervine abordarea "Orchestrării Inteligente a Fluxurilor de Lucru". Prin utilizarea componentelor de IA, dezvoltatorii pot crea fluxuri de lucru care sunt inteligente, adaptive și care se auto-optimizează. IA poate analiza cantități vaste de date, poate învăța din experiențele anterioare și poate lua decizii informate în timp real pentru a orchestra eficient fluxul de lucru.

Beneficii Cheie

1. **Eficiență Crescută**: IA poate optimiza alocarea sarcinilor, utilizarea resurselor și execuția fluxului de lucru, ducând la timpi de procesare mai rapizi și o eficiență generală îmbunătățită.
2. **Adaptabilitate**: Fluxurile de lucru bazate pe IA se pot adapta dinamic la condițiile în schimbare, cum ar fi fluctuațiile în cererea utilizatorilor, performanța sistemului sau cerințele de business, asigurând că aplicația rămâne reactivă și rezilientă.
3. **Luare Automată a Deciziilor**: IA poate automatiza procesele complexe de luare a deciziilor în cadrul fluxului de lucru, reducând intervenția manuală și minimizând riscul erorilor umane.
4. **Personalizare**: IA poate analiza comportamentul, preferințele și contextul utilizatorului pentru a personaliza fluxul de lucru și a oferi experiențe adaptate fiecărui utilizator în parte.
5. **Scalabilitate**: Fluxurile de lucru bazate pe IA pot scala fără probleme pentru a gestiona volume crescânde de date și interacțiuni ale utilizatorilor, fără a compromite performanța sau fiabilitatea.

În secțiunile următoare, vom explora tiparele și tehnicile cheie care permit implementarea fluxurilor de lucru inteligente și vom prezenta exemple din lumea reală despre modul în care IA transformă gestionarea fluxurilor de lucru în aplicațiile moderne.

Tipare Cheie

Pentru a implementa orchestrarea inteligentă a fluxurilor de lucru în aplicații, dezvoltatorii pot utiliza mai multe tipare cheie care valorifică puterea IA. Aceste tipare oferă o abordare structurată pentru proiectarea și gestionarea fluxurilor de lucru, permițând aplicațiilor să se adapteze, să optimizeze și să automatizeze procesele pe baza datelor și contextului în timp real. Să explorăm câteva dintre tiparele fundamentale în orchestrarea inteligentă a fluxurilor de lucru.

Direcționarea Dinamică a Sarcinilor

Acest tipar implică utilizarea IA pentru a dirija inteligent sarcinile într-un flux de lucru pe baza diferiților factori, cum ar fi prioritatea sarcinii, disponibilitatea resurselor și performanța sistemului. Algoritmii de IA pot analiza caracteristicile fiecărei sarcini, lua în considerare starea curentă a sistemului și lua decizii informate pentru a atribui sarcini resurselor sau căilor de procesare cele mai adecvate. Direcționarea dinamică a sarcinilor asigură că sarcinile sunt distribuite și executate eficient, optimizând performanța generală a fluxului de lucru.

```
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
```

Observați bucla creată de expresia `while` de pe linia 29, care continuă să solicite răspunsuri de la AI până când sarcina este atribuită. Pe linia 35, salvăm transcriptul sarcinii pentru analiză și depanare ulterioară, dacă acest lucru devine necesar.

Luarea Deciziilor Contextuale

Puteți utiliza cod foarte similar pentru a lua decizii bazate pe context în cadrul unui workflow. Prin analizarea punctelor de date relevante, cum ar fi preferințele utilizatorului, tiparele istorice și datele în timp real, componentele AI pot determina cel mai potrivit curs al acțiunii la fiecare punct de decizie din workflow. Adaptați comportamentul workflow-ului în funcție de contextul specific al fiecărui utilizator sau scenariu, oferind experiențe personalizate și optimizate.

Compoziţia Adaptivă a Workflow-ului

Acest model se concentrează pe compunerea şi ajustarea dinamică a workflow-urilor în funcţie de cerinţele sau condiţiile în schimbare. AI-ul poate analiza starea curentă a workflow-ului, identifica blocajele sau ineficienţele şi modifica automat structura workflow-ului pentru a optimiza performanţa. Compoziţia adaptivă a workflow-ului permite aplicaţiilor să evolueze şi să-şi îmbunătăţească continuu procesele fără a necesita intervenţie manuală.

Gestionarea şi Recuperarea după Excepţii

Gestionarea şi recuperarea după excepţii sunt aspecte critice ale orchestrării inteligente a workflow-urilor. Când lucraţi cu componente AI şi workflow-uri complexe, este esenţial să anticipaţi şi să gestionaţi excepţiile în mod elegant pentru a asigura stabilitatea şi fiabilitatea sistemului.

Iată câteva consideraţii şi tehnici cheie pentru gestionarea şi recuperarea după excepţii în workflow-urile inteligente:

1. **Propagarea Excepţiilor**: Implementaţi o abordare consistentă pentru propagarea excepţiilor între componentele workflow-ului. Când apare o excepţie într-o componentă, aceasta ar trebui să fie captată, înregistrată şi propagată către orchestrator sau către o componentă discretă responsabilă de gestionarea excepţiilor. Ideea este de a centraliza gestionarea excepţiilor şi de a preveni pierderea silenţioasă a acestora, precum şi de a deschide posibilităţi pentru Gestionarea Inteligentă a Erorilor.
2. **Mecanisme de Reîncercare**: Mecanismele de reîncercare ajută la îmbunătăţirea rezilienţei workflow-ului şi la gestionarea elegantă a eşecurilor intermitente. Este recomandabil să implementaţi mecanisme de reîncercare pentru excepţiile tranzitorii sau recuperabile, cum ar fi probleme de conectivitate la reţea sau

indisponibilitatea resurselor care pot fi reîncercate automat după o întârziere specificată. Având un orchestrator sau un handler de excepții bazat pe AI înseamnă că strategiile de reîncercare nu trebuie să fie mecanice, bazându-se pe algoritmi ficși precum fallback-ul exponențial. Puteți lăsa gestionarea reîncercării la "discreția" componentei AI responsabile de decizia modului de tratare a excepției.

3. **Strategii Alternative**: Dacă o componentă AI eșuează în a furniza un răspuns valid sau întâmpină o eroare—o situație frecventă având în vedere natura sa de ultimă generație—aveți pregătit un mecanism alternativ pentru a asigura continuitatea workflow-ului. Aceasta ar putea implica utilizarea valorilor implicite, algoritmi alternativi sau un Om în Buclă pentru a lua decizii și a menține workflow-ul în mișcare.
4. **Acțiuni Compensatorii**: Directivele orchestratorului ar trebui să includă instrucțiuni despre acțiunile compensatorii pentru gestionarea excepțiilor care nu pot fi rezolvate automat. Acțiunile compensatorii sunt pași întreprinși pentru a anula sau atenua efectele unei operațiuni eșuate. De exemplu, dacă un pas de procesare a plății eșuează, o acțiune compensatorie ar putea fi anularea tranzacției și notificarea utilizatorului. Acțiunile compensatorii ajută la menținerea consistenței și integrității datelor în fața excepțiilor.
5. **Monitorizarea și Alertarea Excepțiilor**: Configurați mecanisme de monitorizare și alertare pentru a detecta și notifica părțile interesate relevante despre excepțiile critice. Orchestratorul poate fi configurat cu praguri și reguli pentru a declanșa alerte când excepțiile depășesc anumite limite sau când apar anumite tipuri de excepții. Acest lucru permite identificarea și rezolvarea proactivă a problemelor înainte ca acestea să afecteze sistemul general.

Iată un exemplu de gestionare și recuperare după excepții într-o componentă workflow în Ruby:

```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
```

În acest exemplu, componenta `InventoryManager` verifică disponibilitatea unui produs pentru o comandă dată. Dacă cantitatea disponibilă este insuficientă, aceasta generează o eroare `InsufficientInventoryError`. Excepția este captată, înregistrată și se implementează un mecanism de reîncercare. Dacă limita de reîncercări este depășită, componenta recurge la intervenția manuală prin notificarea unui administrator.

Prin implementarea unor mecanisme robuste de gestionare și recuperare a excepțiilor, puteți asigura că fluxurile dvs. de lucru inteligente sunt reziliente, ușor de întreținut și

capabile să gestioneze situațiile neașteptate într-un mod elegant.

Aceste modele formează baza orchestrării fluxurilor de lucru inteligente și pot fi combinate și adaptate pentru a satisface cerințele specifice ale diferitelor aplicații. Prin utilizarea acestor modele, dezvoltatorii pot crea fluxuri de lucru care sunt flexibile, reziliente și optimizate pentru performanță și experiența utilizatorului.

În secțiunea următoare, vom explora modul în care aceste modele pot fi implementate în practică, folosind exemple din lumea reală și fragmente de cod pentru a ilustra integrarea componentelor AI în gestionarea fluxurilor de lucru.

Implementarea Orchestrării Fluxurilor de Lucru Inteligente în Practică

Acum că am explorat modelele cheie în orchestrarea fluxurilor de lucru inteligente, să ne cufundăm în modul în care aceste modele pot fi implementate în aplicații din lumea reală. Vom oferi exemple practice și fragmente de cod pentru a ilustra integrarea componentelor AI în gestionarea fluxurilor de lucru.

Procesator Inteligent de Comenzi

Să analizăm un exemplu practic de implementare a orchestrării fluxurilor de lucru inteligente folosind o componentă `OrderProcessor` bazată pe AI într-o aplicație de e-commerce în Ruby on Rails. `OrderProcessor` realizează conceptul de Process Manager Enterprise Integration pe care l-am întâlnit prima dată în Capitolul 3 când am discutat despre Multitudine de Lucrători. Componenta va fi responsabilă pentru gestionarea fluxului de procesare a comenzilor, luarea deciziilor de rutare bazate pe rezultate intermediare și orchestrarea execuției diferitelor etape de procesare.

Procesul de îndeplinire a comenzilor implică mai mulţi paşi, cum ar fi validarea comenzii, verificarea stocului, procesarea plăţii şi expedierea. Fiecare pas este implementat ca un proces lucrător separat care efectuează o sarcină specifică şi returnează rezultatul către `OrderProcessor`. Paşii nu sunt obligatorii şi nici măcar nu trebuie neapărat să fie efectuaţi într-o ordine precisă.

Iată un exemplu de implementare a `OrderProcessor`. Acesta include două mixinuri din Raix. Primul (`ChatCompletion`) îi oferă capacitatea de a face chat completion, ceea ce face din aceasta o componentă AI. Al doilea (`FunctionDispatch`) permite apelarea funcţiilor de către AI, permiţându-i să răspundă la un prompt cu o invocare de funcţie în loc de un mesaj text.

Funcţiile lucrătoare (`validate_order`, `check_inventory`, etc.) delegă către clasele lor respective de lucrători, care pot fi componente AI sau non-AI, cu singura cerinţă fiind ca acestea să returneze rezultatele muncii lor într-un format care poate fi reprezentat ca şir de caractere.

Ca şi în cazul tuturor celorlalte exemple din această parte a cărţii, acest cod este practic pseudo-cod şi este menit doar să transmită semnificaţia modelului şi să inspire propriile dvs. creaţii. Descrierile complete ale modelelor şi exemplele complete de cod sunt incluse în Partea 2.

```ruby
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
```

```
43
44    def process_payment
45      PaymentProcessingWorker.perform(@order)
46    end
47
48    def schedule_shipping
49      ShippingSchedulerWorker.perform(@order)
50    end
51
52    def send_confirmation
53      OrderConfirmationWorker.perform(@order)
54    end
55
56    def finished_processing
57      @order.update!(transcript:, processed_at: Time.current)
58      stop_looping!
59    end
60  end
```

În exemplu, OrderProcessor este inițializat cu un obiect de comandă și menține o transcriere a execuției fluxului de lucru, în formatul tipic de transcriere a conversației care este nativ modelelor de limbaj de mari dimensiuni. Controlul complet este dat AI-ului pentru a orchestra execuția diferitelor etape de procesare, cum ar fi validarea comenzii, verificarea stocului, procesarea plății și livrarea.

De fiecare dată când metoda `chat_completion` este apelată, transcrierea este trimisă către AI pentru a furniza o completare sub forma unui apel de funcție. Este în întregime la latitudinea AI-ului să analizeze rezultatul pasului anterior și să determine acțiunea adecvată de urmat. De exemplu, dacă verificarea stocului relevă nivele scăzute, `OrderProcessor` poate programa o sarcină de reaprovizionare. Dacă procesarea plății eșuează, poate iniția o reîncercare sau poate notifica serviciul de asistență clienți.

> Exemplul de mai sus nu are funcții definite pentru reaprovizionare sau notificarea asistenței clienți, dar ar putea avea cu siguranță.

Transcrierea crește de fiecare dată când o funcție este apelată și servește ca înregistrare a execuției fluxului de lucru, incluzând rezultatele fiecărui pas și instrucțiunile generate de AI pentru pașii următori. Această transcriere poate fi folosită pentru depanare, auditare și oferirea vizibilității în procesul de îndeplinire a comenzii.

Prin utilizarea AI-ului în `OrderProcessor`, aplicația de e-commerce poate adapta dinamic fluxul de lucru bazat pe date în timp real și poate gestiona excepțiile în mod inteligent. Componenta AI poate lua decizii informate, optimiza fluxul de lucru și asigura procesarea lină a comenzilor chiar și în scenarii complexe.

Faptul că singura cerință pentru procesele de execuție este să returneze o ieșire inteligibilă pe care AI-ul să o ia în considerare când decide ce să facă în continuare, ar putea începe să vă facă să realizați cum această abordare poate reduce munca de mapare intrare/ieșire care este de obicei implicată în integrarea sistemelor disparate între ele.

Moderator Inteligent de Conținut

Aplicațiile de social media necesită în general cel puțin o moderare minimală a conținutului pentru a asigura o comunitate sigură și sănătoasă. Acest exemplu de componentă `ContentModerator` folosește AI-ul pentru a orchestra inteligent fluxul de moderare, luând decizii bazate pe caracteristicile conținutului și rezultatele diferitelor etape de moderare.

Procesul de moderare implică mai mulți pași, cum ar fi analiza textului, recunoașterea imaginilor, evaluarea reputației utilizatorului și revizuirea manuală. Fiecare pas este implementat ca un proces de execuție separat care efectuează o sarcină specifică și returnează rezultatul către `ContentModerator`.

Iată un exemplu de implementare a `ContentModerator`:

```
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
```

```
83
84    def reject_content
85      @content.update!(status: 'rejected', transcript: @transcript)
86    end
87
88    private
89
90    def continue_with(result)
91      @transcript << { function: result }
92      complete(@transcript)
93    end
94  end
```

În acest exemplu, `ContentModerator` este inițializat cu un obiect de conținut și menține un transcript de moderare în format conversațional. Componenta AI are control complet asupra fluxului de moderare, decidând ce pași să execute în funcție de caracteristicile conținutului și rezultatele fiecărui pas.

Funcțiile worker disponibile pe care AI-ul le poate invoca includ `analyze_text`, `recognize_image`, `assess_user_reputation` și `escalate_to_manual_review`. Fiecare funcție delegă sarcina către un proces worker corespunzător (`TextAnalysisWorker`, `ImageRecognitionWorker`, etc.) și adaugă rezultatul la transcriptul de moderare, cu excepția funcției de escaladare, care acționează ca o stare finală. În cele din urmă, funcțiile `approve_content` și `reject_content` acționează, de asemenea, ca stări finale.

Componenta AI analizează conținutul și determină acțiunea adecvată de luat. Dacă conținutul conține referințe la imagini, poate apela worker-ul `recognize_image` pentru asistență în evaluarea vizuală. Dacă vreun worker avertizează despre conținut potențial dăunător, AI-ul poate decide să escaladeze conținutul pentru revizuire manuală sau pur și simplu să-l respingă direct. Dar în funcție de severitatea avertismentului, AI-ul poate alege să utilizeze rezultatele evaluării reputației utilizatorului în deciderea modului de gestionare a conținutului despre care nu este sigur. În funcție de cazul de utilizare, poate utilizatorii de încredere au mai multă libertate în ceea ce pot posta. Și așa mai departe...

La fel ca în exemplul anterior cu managerul de procese, transcriptul de moderare servește ca înregistrare a execuției fluxului de lucru, incluzând rezultatele fiecărui pas și deciziile generate de AI. Acest transcript poate fi utilizat pentru auditare, transparență și îmbunătățirea procesului de moderare în timp.

Prin utilizarea AI-ului în `ContentModerator`, aplicația de social media poate adapta dinamic fluxul de moderare bazat pe caracteristicile conținutului și poate gestiona inteligent scenarii complexe de moderare. Componenta AI poate lua decizii informate, optimiza fluxul de lucru și asigura o experiență comunitară sigură și sănătoasă.

Să explorăm încă două exemple care demonstrează planificarea predictivă a sarcinilor și gestionarea și recuperarea excepțiilor în contextul orchestrării inteligente a fluxurilor de lucru.

Planificarea Predictivă a Sarcinilor într-un Sistem de Suport pentru Clienţi

Într-o aplicație de suport pentru clienți construită cu Ruby on Rails, gestionarea și prioritizarea eficientă a tichetelor de suport este crucială pentru oferirea de asistență în timp util clienților. Componenta `SupportTicketScheduler` folosește AI-ul pentru a planifica și atribui predictiv tichete de suport agenților disponibili pe baza diferiților factori, cum ar fi urgența tichetului, expertiza agentului și volumul de muncă.

```
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
```

În acest exemplu, `SupportTicketScheduler` este inițializat cu un obiect de tichet de asistență și menține un transcript de planificare. Componenta AI analizează detaliile tichetului și planifică predictiv atribuirea tichetului pe baza factorilor precum urgența tichetului, expertiza agentului și volumul de muncă prezis al agentului.

Funcțiile disponibile pe care AI-ul le poate invoca includ `analyze_ticket_urgency`, `list_available_agents`, `predict_agent_workload` și `assign_ticket_to_agent`. Fiecare funcție delegă sarcina către o componentă corespunzătoare de analiză sau predicție și adaugă rezultatul la transcriptul de planificare. AI-ul are, de asemenea, opțiunea de a întârzia atribuirea folosind funcția `delay_assignment`.

Componenta AI examinează transcriptul de planificare și ia decizii informate privind atribuirea tichetelor. Aceasta ia în considerare urgența tichetului, expertiza agenților disponibili și volumul de muncă prezis pentru fiecare agent pentru a determina agentul cel mai potrivit pentru gestionarea tichetului.

Prin utilizarea planificării predictive a sarcinilor, aplicația de asistență pentru clienți poate optimiza atribuirea tichetelor, reduce timpii de răspuns și îmbunătăți satisfacția generală a clienților. Gestionarea proactivă și eficientă a tichetelor de asistență asigură că tichetele potrivite sunt atribuite agenților potriviți la momentul potrivit.

Gestionarea Excepţiilor şi Recuperarea într-un Pipeline de Procesare a Datelor

Gestionarea excepțiilor și recuperarea după eșecuri sunt esențiale pentru asigurarea integrității datelor și prevenirea pierderii datelor. Componenta `DataProcessingOrchestrator` utilizează AI pentru a gestiona inteligent excepțiile și pentru a orchestra procesul de recuperare într-un pipeline de procesare a datelor

```
class DataProcessingOrchestrator
  include Raix::ChatCompletion
  include Raix::FunctionDispatch

  SYSTEM_DIRECTIVE = "You are a data processing orchestrator..."

  def initialize(data_batch)
    @data_batch = data_batch
    @transcript = [
      { system: SYSTEM_DIRECTIVE },
      { user: data_batch.to_json }
    ]
  end

  def perform
    complete(@transcript)
  end

  def model
    "openai/gpt-4"
  end

  def functions
    [
      {
        name: "validate_data",
        # ...
      },
      {
        name: "process_data",
        # ...
      },
      {
        name: "request_fix",
        # ...
      },
      {
        name: "retry_processing",
        # ...
      },
      {
        name: "mark_data_as_failed",
```

```
43          # ...
44        },
45        {
46          name: "finished",
47          # ...
48        }
49      ]
50    end
51
52    # implementation of functions that can be called by the AI
53    # entirely at its discretion, depending on the needs of the order
54
55    def validate_data
56      result = DataValidator.perform(@data_batch)
57      continue_with(result)
58    rescue ValidationException => e
59      handle_validation_exception(e)
60    end
61
62    def process_data
63      result = DataProcessor.perform(@data_batch)
64      continue_with(result)
65    rescue ProcessingException => e
66      handle_processing_exception(e)
67    end
68
69    def request_fix(description_of_fix)
70      result = SmartDataFixer.new(description_of_fix, @data_batch)
71      continue_with(result)
72    end
73
74    def retry_processing(timeout_in_seconds)
75      wait(timeout_in_seconds)
76      process_data
77    end
78
79    def mark_data_as_failed
80      @data_batch.update!(status: 'failed', transcript: @transcript)
81    end
82
83    def finished
84      @data_batch.update!(status: 'finished', transcript: @transcript)
```

```
  end

  private

  def continue_with(result)
    @transcript << { function: result }
    complete(@transcript)
  end

  def handle_validation_exception(exception)
    @transcript << { exception: exception.message }
    complete(@transcript)
  end

  def handle_processing_exception(exception)
    @transcript << { exception: exception.message }
    complete(@transcript)
  end
end
```

În acest exemplu, `DataProcessingOrchestrator` este inițializat cu un obiect de date în lot și menține o transcriere a procesării. Componenta AI orchestrează pipeline-ul de procesare a datelor, gestionând excepțiile și recuperând din eșecuri după necesitate.

Funcțiile disponibile pentru AI spre invocare includ `validate_data`, `process_data`, `request_fix`, `retry_processing`, și `mark_data_as_failed`. Fiecare funcție delegă sarcina către o componentă corespunzătoare de procesare a datelor și adaugă rezultatul sau detaliile excepției la transcrierea procesării.

Dacă o excepție de validare apare în timpul etapei `validate_data`, funcția `handle_validation_exception` adaugă datele excepției la transcriere și returnează controlul către AI. În mod similar, dacă o excepție de procesare apare în timpul etapei `process_data`, AI-ul poate decide strategia de recuperare.

În funcție de natura excepției întâlnite, AI-ul poate decide la discreția sa să apeleze `request_fix`, care delegă către o componentă AI `SmartDataFixer` (vezi capitolul despre Date cu Auto-reparare). Reparatorul de date primește o descriere în limbaj

natural despre cum ar trebui să modifice `@data_batch` astfel încât procesarea să poată fi reîncercată. Poate că o reîncercare cu succes ar implica eliminarea înregistrărilor din lotul de date care nu au trecut validarea și/sau copierea lor într-un pipeline de procesare diferit pentru revizuire umană? Posibilitățile sunt aproape nelimitate.

Prin încorporarea gestionării și recuperării excepțiilor bazate pe AI, aplicația de procesare a datelor devine mai rezistentă și tolerantă la erori. `DataProcessingOrchestrator` gestionează inteligent excepțiile, minimizează pierderea datelor și asigură execuția fluidă a fluxului de lucru de procesare a datelor.

Monitorizare şi Jurnalizare

Monitorizarea și jurnalizarea oferă vizibilitate asupra progresului, performanței și sănătății componentelor fluxului de lucru bazat pe AI, permițând dezvoltatorilor să urmărească și să analizeze comportamentul sistemului. Implementarea mecanismelor eficiente de monitorizare și jurnalizare este esențială pentru depanare, auditare și îmbunătățirea continuă a fluxurilor de lucru inteligente.

Monitorizarea Progresului şi Performanţei Fluxului de Lucru

Pentru a asigura execuția fluidă a fluxurilor de lucru inteligente, este important să monitorizăm progresul și performanța fiecărei componente a fluxului de lucru. Aceasta implică urmărirea metricilor și evenimentelor cheie pe parcursul ciclului de viață al fluxului de lucru.

Câteva aspecte importante de monitorizat includ:

1. **Timpul de Execuție al Fluxului de Lucru**: Măsurarea timpului necesar fiecărei componente a fluxului de lucru pentru a-și finaliza sarcina. Aceasta ajută la identificarea blocajelor de performanță și optimizarea eficienței generale a fluxului de lucru.

2. **Utilizarea Resurselor**: Monitorizarea utilizării resurselor sistemului, cum ar fi CPU, memoria și stocarea, de către fiecare componentă a fluxului de lucru. Aceasta ajută la asigurarea că sistemul operează în limitele capacității sale și poate gestiona eficient volumul de muncă.

3. **Rate de Eroare și Excepții**: Urmărirea apariției erorilor și excepțiilor în cadrul componentelor fluxului de lucru. Aceasta ajută la identificarea potențialelor probleme și permite gestionarea proactivă a erorilor și recuperarea.

4. **Puncte de Decizie și Rezultate**: Monitorizarea punctelor de decizie din fluxul de lucru și a rezultatelor deciziilor bazate pe AI. Aceasta oferă perspective asupra comportamentului și eficacității componentelor AI.

Datele capturate prin procesele de monitorizare pot fi afișate în tablouri de bord sau utilizate ca intrări pentru rapoarte programate care informează administratorii de sistem despre sănătatea sistemului.

Datele de monitorizare pot fi transmise unui proces de administrator de sistem bazat pe AI pentru revizuire și potențială acțiune!

Jurnalizarea Evenimentelor şi Deciziilor Cheie

Jurnalizarea este o practică esențială care implică capturarea și stocarea informațiilor relevante despre evenimente cheie, decizii și excepții care apar în timpul execuției fluxului de lucru.

Câteva aspecte importante de jurnalizat includ:

1. **Inițierea și Finalizarea Fluxului de Lucru**: Jurnalizarea timpilor de început și sfârșit ai fiecărei instanțe a fluxului de lucru, împreună cu orice metadate relevante precum datele de intrare și contextul utilizatorului.

2. **Execuția Componentelor**: Jurnalizarea detaliilor de execuție ale fiecărei componente a fluxului de lucru, inclusiv parametrii de intrare, rezultatele de ieșire și orice date intermediare generate.

3. **Deciziile AI și Raționamentul**: Jurnalizarea deciziilor luate de componentele AI, împreună cu raționamentul subiacent sau scorurile de încredere. Aceasta oferă transparență și permite auditarea deciziilor bazate pe AI.

4. **Excepții și Mesaje de Eroare**: Jurnalizarea oricăror excepții sau mesaje de eroare întâlnite în timpul execuției fluxului de lucru, inclusiv urma stivei și informațiile contextuale relevante.

Jurnalizarea poate fi implementată folosind diverse tehnici, cum ar fi scrierea în fișiere de jurnal, stocarea jurnalelor într-o bază de date sau trimiterea jurnalelor către un serviciu centralizat de jurnalizare. Este important să se aleagă un framework de jurnalizare care oferă flexibilitate, scalabilitate și integrare ușoară cu arhitectura aplicației.

Iată un exemplu despre cum poate fi implementată jurnalizarea într-o aplicație Ruby on Rails folosind clasa `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)
```

Prin plasarea strategică a instrucțiunilor de jurnalizare în cadrul componentelor fluxului

de lucru şi a punctelor de decizie AI, dezvoltatorii pot captura informaţii valoroase pentru depanare, auditare şi analiză.

Beneficiile Monitorizării şi Jurnalizării

Implementarea monitorizării şi jurnalizării în orchestrarea fluxurilor de lucru inteligente oferă mai multe beneficii:

1. **Depanare şi Rezolvarea Problemelor**: Jurnalele detaliate şi datele de monitorizare ajută dezvoltatorii să identifice şi să diagnosticheze rapid problemele. Acestea oferă informaţii despre fluxul de execuţie al procesului, interacţiunile dintre componente şi orice erori sau excepţii întâlnite.

2. **Optimizarea Performanţei**: Monitorizarea metricilor de performanţă permite dezvoltatorilor să identifice blocajele şi să optimizeze componentele fluxului de lucru pentru o eficienţă mai bună. Prin analizarea timpilor de execuţie, a utilizării resurselor şi a altor metrici, dezvoltatorii pot lua decizii informate pentru a îmbunătăţi performanţa generală a sistemului.

3. **Auditare şi Conformitate**: Jurnalizarea evenimentelor şi deciziilor cheie oferă o pistă de audit pentru conformitatea cu reglementările şi responsabilitatea. Aceasta permite organizaţiilor să urmărească şi să verifice acţiunile efectuate de componentele AI şi să asigure respectarea regulilor de afaceri şi a cerinţelor legale.

4. **Îmbunătăţire Continuă**: Datele de monitorizare şi jurnalizare servesc ca informaţii valoroase pentru îmbunătăţirea continuă a fluxurilor de lucru inteligente. Prin analizarea datelor istorice, identificarea tiparelor şi măsurarea eficacităţii deciziilor AI, dezvoltatorii pot rafina şi îmbunătăţi iterativ logica de orchestrare a fluxului de lucru.

Consideraţii şi Bune Practici

La implementarea monitorizării şi jurnalizării în orchestrarea fluxurilor de lucru inteligente, luaţi în considerare următoarele bune practici:

1. **Definiți Metrici Clare de Monitorizare**: Identificați metricile și evenimentele cheie care trebuie monitorizate în funcție de cerințele specifice ale fluxului de lucru. Concentrați-vă pe metrici care oferă informații semnificative despre performanța, starea și comportamentul sistemului.

2. **Implementați Jurnalizare Granulară**: Asigurați-vă că instrucțiunile de jurnalizare sunt plasate în puncte adecvate în cadrul componentelor fluxului de lucru și a punctelor de decizie AI. Capturați informații contextuale relevante, cum ar fi parametrii de intrare, rezultatele și orice date intermediare generate.

3. **Utilizați Jurnalizare Structurată**: Adoptați un format de jurnalizare structurat pentru a facilita analiza și procesarea datelor din jurnal. Jurnalizarea structurată permite o mai bună căutare, filtrare și agregare a înregistrărilor din jurnal.

4. **Gestionați Retenția și Rotația Jurnalelor**: Implementați politici de retenție și rotație a jurnalelor pentru a gestiona stocarea și ciclul de viață al fișierelor jurnal. Determinați perioada de retenție adecvată în funcție de cerințele legale, constrângerile de stocare și nevoile de analiză. Dacă este posibil, externalizați jurnalizarea către un serviciu terț precum Papertrail.

5. **Securizați Informațiile Sensibile**: Fiți prudenți când jurnalizați informații sensibile, cum ar fi informațiile de identificare personală (PII) sau date confidențiale de afaceri. Implementați măsuri de securitate adecvate, cum ar fi mascarea datelor sau criptarea, pentru a proteja informațiile sensibile în fișierele jurnal.

6. **Integrați cu Instrumente de Monitorizare și Alertare**: Folosiți instrumente de monitorizare și alertare pentru centralizarea colectării, analizei și vizualizării datelor de monitorizare și jurnalizare. Aceste instrumente pot oferi informații în timp real, pot genera alerte bazate pe praguri predefinite și pot facilita detectarea și rezolvarea proactivă a problemelor. Instrumentul meu preferat dintre acestea este Datadog.

Prin implementarea unor mecanisme complete de monitorizare și jurnalizare, dezvoltatorii pot obține informații valoroase despre comportamentul și performanța fluxurilor de lucru inteligente. Aceste informații permit depanarea eficientă,

optimizarea și îmbunătățirea continuă a sistemelor de orchestrare a fluxurilor de lucru bazate pe AI.

Consideraţii privind Scalabilitatea şi Performanţa

Scalabilitatea și performanța sunt aspecte critice de luat în considerare la proiectarea și implementarea sistemelor de orchestrare a fluxurilor de lucru inteligente. Pe măsură ce volumul fluxurilor de lucru concurente și complexitatea componentelor bazate pe AI cresc, devine esențial să ne asigurăm că sistemul poate gestiona eficient încărcarea și poate scala fără probleme pentru a satisface cerințele în creștere.

Gestionarea Volumelor Mari de Fluxuri de Lucru Concurente

Sistemele de orchestrare a fluxurilor de lucru inteligente trebuie adesea să gestioneze un număr mare de fluxuri de lucru concurente. Pentru a asigura scalabilitatea, luaţi în considerare următoarele strategii:

1. **Procesare Asincronă**: Implementaţi mecanisme de procesare asincronă pentru a decupla execuția componentelor fluxului de lucru. Acest lucru permite sistemului să gestioneze multiple fluxuri de lucru concurente fără a bloca sau aștepta finalizarea fiecărei componente. Procesarea asincronă poate fi realizată folosind cozi de mesaje, arhitecturi bazate pe evenimente sau framework-uri de procesare a job-urilor în fundal precum Sidekiq.

2. **Arhitectură Distribuită**: Proiectaţi arhitectura sistemului pentru a utiliza componente serverless (precum AWS Lambda) sau pur și simplu distribuiţi încărcarea între mai multe noduri sau servere alături de serverul principal al aplicației. Acest

lucru permite scalabilitate orizontală, unde noduri suplimentare pot fi adăugate pentru a gestiona volume crescute de fluxuri de lucru.

3. **Execuție Paralelă**: Identificați oportunități pentru execuție paralelă în cadrul fluxurilor de lucru. Unele componente ale fluxului de lucru pot fi independente una de alta și pot fi executate concurent. Prin utilizarea tehnicilor de procesare paralelă, cum ar fi multi-threading sau cozi de sarcini distribuite, sistemul poate optimiza utilizarea resurselor și reduce timpul total de execuție al fluxului de lucru.

Optimizarea Performanţei Componentelor Bazate pe IA

Componentele bazate pe Inteligență Artificială, precum modelele de învățare automată sau motoarele de procesare a limbajului natural, pot fi intensive din punct de vedere computațional și pot afecta performanța generală a sistemului de orchestrare a fluxurilor de lucru. Pentru a optimiza performanța componentelor IA, luați în considerare următoarele tehnici:

1. **Memorarea în Cache**: Dacă procesarea IA este pur generativă și nu implică căutări de informații în timp real sau integrări externe pentru a genera completările de chat, atunci puteţi explora mecanisme de memorare în cache pentru a stoca și reutiliza rezultatele operațiilor frecvent accesate sau costisitoare din punct de vedere computațional.

2. **Optimizarea Modelului**: Optimizați continuu modul în care utilizați modelele IA în componentele fluxului de lucru. Aceasta poate implica tehnici precum *Distilarea Prompturilor* sau poate fi doar o chestiune de testare a noilor modele pe măsură ce acestea devin disponibile.

3. **Procesarea în Loturi**: Dacă lucrați cu modele de clasă GPT-4, este posibil să puteţi folosi tehnici de procesare în loturi pentru a procesa mai multe puncte de date sau cereri într-un singur lot, în loc să le procesați individual. Prin procesarea datelor în loturi, sistemul poate optimiza utilizarea resurselor și reduce supraîncărcarea cauzată de cererile repetate către model.

Monitorizarea şi Profilarea Performanţei

Pentru a identifica blocajele de performanță și a optimiza scalabilitatea sistemului inteligent de orchestrare a fluxurilor de lucru, este crucial să implementați mecanisme de monitorizare și profilare. Luați în considerare următoarele abordări:

1. **Metrici de Performanță**: Definiți și urmăriți metrici cheie de performanță, cum ar fi timpul de răspuns, capacitatea de procesare, utilizarea resurselor și latența. Aceste metrici oferă informații despre performanța sistemului și ajută la identificarea zonelor care necesită optimizare. Agregatorul popular de modele IA OpenRouter include metricile Host[1] și Speed[2] în fiecare răspuns API, făcând trivială urmărirea acestor metrici cheie.

2. **Instrumente de Profilare**: Utilizați instrumente de profilare pentru a analiza performanța componentelor individuale ale fluxului de lucru și a operațiilor IA. Instrumentele de profilare pot ajuta la identificarea punctelor fierbinți de performanță, a căilor de cod ineficiente sau a operațiilor care consumă multe resurse. Instrumentele populare de profilare includ New Relic, Scout, sau profilere integrate furnizate de limbajul de programare sau framework.

3. **Testarea sub Sarcină**: Efectuați teste sub sarcină pentru a evalua performanța sistemului în condiții diferite de încărcare concurentă. Testarea sub sarcină ajută la identificarea limitelor de scalabilitate ale sistemului, detectarea degradării performanței și asigurarea că sistemul poate gestiona traficul așteptat fără a compromite performanța.

4. **Monitorizare Continuă**: Implementați mecanisme de monitorizare continuă și alertare pentru a detecta proactiv probleme și blocaje de performanță. Configurați panouri de monitorizare și alerte pentru a urmări indicatorii cheie de performanță (KPI) și pentru a primi notificări când sunt depășite pragurile predefinite. Acest lucru permite identificarea și rezolvarea promptă a problemelor de performanță.

[1] Host reprezintă timpul necesar pentru a primi primul byte al generării transmise de la gazda modelului, cunoscut și ca "timpul până la primul byte."

[2] Speed este calculată ca numărul de token-uri completate împărțit la timpul total de generare. Pentru cererile netransmise în flux, latența este considerată parte din timpul de generare.

Strategii de Scalare

Pentru a gestiona volumele de lucru în creştere şi a asigura scalabilitatea sistemului inteligent de orchestrare a fluxurilor de lucru, luaţi în considerare următoarele strategii de scalare:

1. **Scalare Verticală**: Scalarea verticală implică creşterea resurselor (de exemplu, CPU, memorie) nodurilor sau serverelor individuale pentru a gestiona volume de lucru mai mari. Această abordare este potrivită când sistemul necesită mai multă putere de procesare sau memorie pentru a gestiona fluxuri de lucru complexe sau operaţii IA.

2. **Scalare Orizontală**: Scalarea orizontală implică adăugarea mai multor noduri sau servere la sistem pentru a distribui volumul de lucru. Această abordare este eficientă când sistemul trebuie să gestioneze un număr mare de fluxuri de lucru concurente sau când volumul de lucru poate fi distribuit uşor între mai multe noduri. Scalarea orizontală necesită o arhitectură distribuită şi mecanisme de echilibrare a încărcării pentru a asigura distribuţia uniformă a traficului.

3. **Auto-Scalare**: Implementaţi mecanisme de auto-scalare pentru a ajusta automat numărul de noduri sau resurse în funcţie de cererea de volum de lucru. Auto-scalarea permite sistemului să se scaleze dinamic în sus sau în jos în funcţie de traficul primit, asigurând utilizarea optimă a resurselor şi eficienţa costurilor. Platforme cloud precum Amazon Web Services (AWS) sau Google Cloud Platform (GCP) oferă capacităţi de auto-scalare care pot fi folosite pentru sisteme inteligente de orchestrare a fluxurilor de lucru.

Tehnici de Optimizare a Performanţei

Pe lângă strategiile de scalare, luaţi în considerare următoarele tehnici de optimizare a performanţei pentru a îmbunătăţi eficienţa sistemului inteligent de orchestrare a fluxurilor de lucru:

1. **Stocarea şi Recuperarea Eficientă a Datelor**: Optimizaţi mecanismele de stocare şi recuperare a datelor utilizate de componentele fluxului de lucru. Utilizaţi indexarea

eficientă a bazelor de date, tehnici de optimizare a interogărilor și memorarea în cache a datelor pentru a minimiza latența și a îmbunătăți performanța operațiilor intensive cu date.

2. **I/O asincron**: Utilizați operațiuni I/O asincrone pentru a preveni blocarea și a îmbunătăți capacitatea de răspuns a sistemului. I/O-ul asincron permite sistemului să gestioneze multiple cereri simultan fără a aștepta finalizarea operațiunilor I/O, maximizând astfel utilizarea resurselor.

3. **Serializare și Deserializare Eficientă**: Optimizați procesele de serializare și deserializare utilizate pentru schimbul de date între componentele fluxului de lucru. Utilizați formate de serializare eficiente, precum Protocol Buffers sau MessagePack, pentru a reduce costurile serializării datelor și a îmbunătăți performanța comunicării între componente.

Pentru aplicațiile bazate pe Ruby, luați în considerare utilizarea Universal ID. Universal ID folosește atât MessagePack cât și Brotli (o combinație construită pentru viteză și compresie de date de top). Când sunt combinate, aceste biblioteci sunt cu până la 30% mai rapide și au rate de compresie cu doar 2-5% mai mici comparativ cu Protocol Buffers.

4. **Compresie și Codificare**: Aplicați tehnici de compresie și codificare pentru a reduce dimensiunea datelor transferate între componentele fluxului de lucru. Algoritmii de compresie, precum gzip sau Brotli, pot reduce semnificativ utilizarea lățimii de bandă a rețelei și pot îmbunătăți performanța generală a sistemului.

Prin luarea în considerare a aspectelor de scalabilitate și performanță în timpul proiectării și implementării sistemelor de orchestrare inteligentă a fluxurilor de lucru, puteți asigura că sistemul dumneavoastră poate gestiona volume mari de fluxuri de lucru concurente, poate optimiza performanța componentelor bazate pe AI și se poate scala fără probleme pentru a satisface cerințele în creștere. Monitorizarea continuă, profilarea și eforturile de optimizare sunt esențiale pentru menținerea performanței și

capacității de răspuns a sistemului pe măsură ce volumul de muncă și complexitatea cresc în timp.

Testarea și Validarea Fluxurilor de Lucru

Testarea și validarea sunt aspecte critice în dezvoltarea și menținerea sistemelor de orchestrare inteligentă a fluxurilor de lucru. Având în vedere natura complexă a fluxurilor de lucru bazate pe AI, este esențial să ne asigurăm că fiecare componentă funcționează conform așteptărilor, că fluxul general de lucru se comportă corect și că deciziile AI sunt precise și de încredere. În această secțiune, vom explora diverse tehnici și considerații pentru testarea și validarea fluxurilor de lucru inteligente.

Testarea Unitară a Componentelor Fluxului de Lucru

Testarea unitară implică testarea componentelor individuale ale fluxului de lucru în izolare pentru a verifica corectitudinea și robustețea acestora. Când testați unitar componente bazate pe AI ale fluxului de lucru, luați în considerare următoarele:

1. **Validarea Datelor de Intrare**: Testați capacitatea componentei de a gestiona diferite tipuri de date de intrare, inclusiv date valide și invalide. Verificați că respectiva componentă gestionează elegant cazurile limită și furnizează mesaje de eroare sau excepții adecvate.

2. **Verificarea Datelor de Ieșire**: Asigurați-vă că componenta produce rezultatul așteptat pentru un set dat de date de intrare. Comparați ieșirea efectivă cu rezultatele așteptate pentru a asigura corectitudinea.

3. **Gestionarea Erorilor**: Testați mecanismele de gestionare a erorilor ale componentei prin simularea diferitelor scenarii de eroare, cum ar fi date de intrare invalide, indisponibilitatea resurselor sau excepții neașteptate. Verificați că componenta captează și gestionează erorile în mod corespunzător.

4. **Condiții Limită**: Testați comportamentul componentei în condiții limită, cum ar fi date de intrare goale, dimensiune maximă a datelor de intrare sau valori extreme. Asigurați-vă că componenta gestionează aceste condiții în mod elegant, fără a se bloca sau a produce rezultate incorecte.

Iată un exemplu de test unitar pentru o componentă a fluxului de lucru în Ruby folosind framework-ul de testare RSpec:

```
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
```

În acest exemplu, componenta `OrderValidator` este testată folosind două cazuri de test: unul pentru o comandă validă și altul pentru o comandă invalidă. Cazurile de test verifică dacă metoda `validate` returnează valoarea booleană așteptată în funcție de validitatea comenzii.

Testarea de Integrare a Interacţiunilor din Fluxul de Lucru

Testarea de integrare se concentrează pe verificarea interacțiunilor și a fluxului de date între diferitele componente ale fluxului de lucru. Aceasta asigură că componentele

funcționează împreună fără probleme și produc rezultatele așteptate. Când testați fluxuri de lucru inteligente, luați în considerare următoarele:

1. **Interacțiunea Componentelor**: Testați comunicarea și schimbul de date între componentele fluxului de lucru. Verificați dacă rezultatul unei componente este transmis corect ca date de intrare către următoarea componentă din fluxul de lucru.

2. **Consistența Datelor**: Asigurați-vă că datele rămân consistente și precise pe măsură ce trec prin fluxul de lucru. Verificați dacă transformările de date, calculele și agregările sunt efectuate corect.

3. **Propagarea Excepțiilor**: Testați modul în care excepțiile și erorile sunt propagate și gestionate între componentele fluxului de lucru. Verificați dacă excepțiile sunt captate, înregistrate și gestionate corespunzător pentru a preveni întreruperea fluxului de lucru.

4. **Comportament Asincron**: Dacă fluxul de lucru implică componente asincrone sau execuție paralelă, testați mecanismele de coordonare și sincronizare. Asigurați-vă că fluxul de lucru se comportă corect în scenarii concurente și asincrone.

Iată un exemplu de test de integrare pentru un flux de lucru în Ruby folosind framework-ul de testare RSpec:

```
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
```

```
17
18 end
```

În acest exemplu, `OrderProcessingWorkflow` este testat prin verificarea interacțiunilor dintre diferitele componente ale fluxului de lucru. Cazul de test stabilește așteptări pentru comportamentul fiecărei componente și asigură că fluxul de lucru procesează comanda cu succes, actualizând în mod corespunzător starea comenzii.

Testarea Punctelor de Decizie AI

Testarea punctelor de decizie AI este crucială pentru a asigura acuratețea și fiabilitatea fluxurilor de lucru bazate pe AI. Când testați punctele de decizie AI, luați în considerare următoarele:

1. **Acuratețea Deciziilor**: Verificați că componenta AI ia decizii precise pe baza datelor de intrare și a modelului antrenat. Comparați deciziile AI cu rezultatele așteptate sau cu datele de referință.

2. **Cazuri Limită**: Testați comportamentul componentei AI în cazuri limită și scenarii neobișnuite. Verificați că componenta AI gestionează aceste cazuri în mod elegant și ia decizii rezonabile.

3. **Prejudecăți și Echitate**: Evaluați componenta AI pentru potențiale prejudecăți și asigurați-vă că ia decizii corecte și nepărtinitoare. Testați componenta cu date de intrare diverse și analizați rezultatele pentru orice tipare discriminatorii.

4. **Explicabilitate**: Dacă componenta AI oferă explicații sau raționamente pentru deciziile sale, verificați corectitudinea și claritatea explicațiilor. Asigurați-vă că explicațiile sunt aliniate cu procesul decizional subiacent.

Iată un exemplu de testare a unui punct de decizie AI în Ruby folosind cadrul de testare RSpec:

```
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
```

În acest exemplu, componenta AI `FraudDetector` este testată cu două cazuri de test: unul pentru o tranzacție frauduloasă și altul pentru o tranzacție legitimă. Cazurile de test verifică dacă metoda `detect_fraud` returnează valoarea booleană așteptată în funcție de caracteristicile tranzacției.

Testarea End-to-End

Testarea end-to-end implică testarea întregului flux de lucru de la început până la sfârșit, simulând scenarii și interacțiuni ale utilizatorilor din lumea reală. Aceasta asigură că fluxul de lucru se comportă corect și produce rezultatele dorite. Când efectuați testare end-to-end pentru fluxuri de lucru inteligente, luați în considerare următoarele:

1. **Scenarii de Utilizare**: Identificați scenariile comune de utilizare și testați comportamentul fluxului de lucru în aceste scenarii. Verificați dacă fluxul de lucru

gestionează corect datele de intrare ale utilizatorului, ia decizii adecvate și produce rezultatele așteptate.

2. **Validarea Datelor**: Asigurați-vă că fluxul de lucru validează și sanitizează datele de intrare ale utilizatorului pentru a preveni inconsistențele de date sau vulnerabilitățile de securitate. Testați fluxul de lucru cu diverse tipuri de date de intrare, inclusiv date valide și invalide.

3. **Recuperarea după Erori**: Testați capacitatea fluxului de lucru de a se recupera după erori și excepții. Simulați scenarii de eroare și verificați dacă fluxul de lucru le gestionează elegant, înregistrează erorile și ia măsurile de recuperare adecvate.

4. **Performanță și Scalabilitate**: Evaluați performanța și scalabilitatea fluxului de lucru în diferite condiții de încărcare. Testați fluxul de lucru cu un volum mare de cereri concurente și măsurați timpii de răspuns, utilizarea resurselor și stabilitatea generală a sistemului.

Iată un exemplu de test end-to-end pentru un flux de lucru în Ruby folosind framework-ul de testare RSpec și biblioteca Capybara pentru simularea interacțiunilor utilizatorului:

```
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
```

În acest exemplu, testul end-to-end simulează un utilizator care plasează o comandă prin intermediul interfeței web. Acesta completează câmpurile necesare din formular,

trimite comanda și verifică dacă aceasta este procesată cu succes, afișând mesajul de confirmare corespunzător și actualizând statusul comenzii în baza de date.

Integrare și Livrare Continuă

Pentru a asigura fiabilitatea și mentenabilitatea fluxurilor de lucru inteligente, se recomandă integrarea testării și validării în pipeline-ul de integrare și livrare continuă (CI/CD). Acest lucru permite testarea și validarea automatizată a modificărilor fluxului de lucru înainte ca acestea să fie implementate în producție. Luați în considerare următoarele practici:

1. **Execuție Automatizată a Testelor**: Configurați pipeline-ul CI/CD pentru a rula automat suita de teste ori de câte ori sunt făcute modificări în codul fluxului de lucru. Acest lucru asigură că orice regresii sau erori sunt detectate devreme în procesul de dezvoltare.

2. **Monitorizarea Acoperirii cu Teste**: Măsurați și monitorizați acoperirea cu teste a componentelor fluxului de lucru și a punctelor de decizie AI. Țintiti spre o acoperire mare cu teste pentru a vă asigura că rutele și scenariile critice sunt testate temeinic.

3. **Feedback Continuu**: Integrați rezultatele testelor și metricile de calitate a codului în fluxul de dezvoltare. Oferiți feedback continuu dezvoltatorilor despre starea testelor, calitatea codului și orice probleme detectate în timpul procesului CI/CD.

4. **Medii de Staging**: Implementați fluxul de lucru în medii de staging care reflectă îndeaproape mediul de producție. Efectuați teste și validări suplimentare în mediul de staging pentru a detecta orice probleme legate de infrastructură, configurare sau integrarea datelor.

5. **Mecanisme de Rollback**: Implementați mecanisme de rollback în cazul eșecurilor de implementare sau a problemelor critice detectate în producție. Asigurați-vă că fluxul de lucru poate fi rapid revenit la o versiune stabilă anterioară pentru a minimiza timpul de

nefuncționare și impactul asupra utilizatorilor.

Prin încorporarea testării și validării pe parcursul ciclului de dezvoltare a fluxurilor de lucru inteligente, organizațiile pot asigura fiabilitatea, acuratețea și mentenabilitatea sistemelor lor bazate pe AI. Testarea și validarea regulată ajută la detectarea bug-urilor, prevenirea regresiilor și construirea încrederii în comportamentul și rezultatele fluxului de lucru.

Partea 2: Tiparele

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Ingineria Prompturilor

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Lanţul de Gândire

Acest conţinut nu este disponibil în cartea de eşantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Cum Funcţionează

Acest conţinut nu este disponibil în cartea de eşantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Exemple

Acest conţinut nu este disponibil în cartea de eşantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Generarea de Conţinut

Acest conţinut nu este disponibil în cartea de eşantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Crearea Entităţilor Structurate

Acest conţinut nu este disponibil în cartea de eşantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Ghidarea Agentului LLM

Acest conţinut nu este disponibil în cartea de eşantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Beneficii şi Consideraţii

Acest conţinut nu este disponibil în cartea de eşantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Comutare de Mod

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Cum Funcționează

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Când să o folosești

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Exemplu

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Atribuirea Rolului

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Cum Funcţionează

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Când să Îl Folosiţi

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Exemple

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Prompt Object

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Cum funcţionează

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Şablon de Prompt

Acest conţinut nu este disponibil în cartea de eşantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Cum funcţionează

Acest conţinut nu este disponibil în cartea de eşantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Beneficii şi considerente

Acest conţinut nu este disponibil în cartea de eşantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Când să îl utilizăm:

Acest conţinut nu este disponibil în cartea de eşantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Exemplu

Acest conţinut nu este disponibil în cartea de eşantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

IO Structurat

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Cum Funcționează

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Scalarea IO-ului Structurat

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Beneficii și Considerații

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Înlănţuirea Prompturilor

Acest conţinut nu este disponibil în cartea de eşantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Cum Funcţionează

Acest conţinut nu este disponibil în cartea de eşantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Când Să Îl Utilizaţi

Acest conţinut nu este disponibil în cartea de eşantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Exemplu: Integrarea Olympia

Acest conţinut nu este disponibil în cartea de eşantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Rescriitor de Prompturi

Acest conţinut nu este disponibil în cartea de eşantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Cum Funcţionează

Acest conţinut nu este disponibil în cartea de eşantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Exemplu

Acest conţinut nu este disponibil în cartea de eşantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Delimitarea Răspunsurilor

Cum Funcţionează

Beneficii şi Consideraţii

Gestionarea Erorilor

Analizorul de Interogări

Acest conţinut nu este disponibil în cartea de eşantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Cum Funcţionează

Acest conţinut nu este disponibil în cartea de eşantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Implementare

Acest conţinut nu este disponibil în cartea de eşantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Etichetarea Părţilor de Vorbire (POS) şi Recunoaşterea Entităţilor Numite (NER)

Acest conţinut nu este disponibil în cartea de eşantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Clasificarea Intenţiei

Acest conţinut nu este disponibil în cartea de eşantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Extragerea Cuvintelor-Cheie

Acest conţinut nu este disponibil în cartea de eşantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Beneficii

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Rescriptor de Interogări

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Cum Funcționează

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Exemplu

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Beneficii

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Ventriloc

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Cum Funcţionează

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Când să-l Utilizaţi

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Exemplu

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Componente Discrete

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Predicat

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Cum Funcţionează

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Când să îl Utilizaţi

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Exemplu

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Faţada API

Acest conţinut nu este disponibil în cartea de eşantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Cum funcţionează

Acest conţinut nu este disponibil în cartea de eşantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Beneficii cheie

Acest conţinut nu este disponibil în cartea de eşantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Când să o foloseşti

Acest conţinut nu este disponibil în cartea de eşantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Exemplu

Acest conţinut nu este disponibil în cartea de eşantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Autentificare şi Autorizare

Acest conţinut nu este disponibil în cartea de eşantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Gestionarea Cererilor

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Formatarea Răspunsurilor

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Gestionarea Erorilor şi Cazuri Limită

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Consideraţii privind Scalabilitatea şi Performanţa

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Comparaţie cu Alte Pattern-uri de Design

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Interpretorul de Rezultate

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Cum Funcționează

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Când să-l Folosești

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Exemplu

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Maşină Virtuală

Acest conţinut nu este disponibil în cartea de eşantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Cum Funcţionează

Acest conţinut nu este disponibil în cartea de eşantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Când să Îl Foloseşti

Acest conţinut nu este disponibil în cartea de eşantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Exemplu

Acest conţinut nu este disponibil în cartea de eşantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

În Spatele Magiei

Acest conţinut nu este disponibil în cartea de eşantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Specificaţie şi Testare

Acest conţinut nu este disponibil în cartea de eşantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Specificarea Comportamentului

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Scrierea Cazurilor de Test

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Exemplu: Testarea Componentei de Traducere

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Reluarea Interacţiunilor HTTP

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Human In The Loop (HITL)

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Modele de Nivel Înalt

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Inteligența Hibridă

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Răspuns Adaptiv

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Schimbarea rolurilor între om şi IA

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Escaladare

Acest conţinut nu este disponibil în cartea de eşantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Cum funcţionează

Acest conţinut nu este disponibil în cartea de eşantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Beneficii cheie

Acest conţinut nu este disponibil în cartea de eşantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Aplicaţie în Lumea Reală: Asistenţa Medicală

Acest conţinut nu este disponibil în cartea de eşantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Bucla de Feedback

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Cum Funcţionează

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Aplicaţii şi Exemple

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Tehnici Avansate în Integrarea Feedbackului Uman

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Radierea Pasivă de Informaţii

Cum Funcţionează

Afişarea Contextuală a Informaţiilor

Notificări Proactive

Explicaţii Interpretative

Explorare Interactivă

Beneficii Cheie

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Aplicaţii şi Exemple

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Luarea Deciziilor în Colaborare (CDM)

Cum Funcţionează

Exemplu

Învăţarea Continuă

Acest conţinut nu este disponibil în cartea de eşantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Cum Funcţionează

Acest conţinut nu este disponibil în cartea de eşantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Aplicaţii şi Exemple

Acest conţinut nu este disponibil în cartea de eşantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Exemplu

Acest conţinut nu este disponibil în cartea de eşantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Consideraţii Etice

Acest conţinut nu este disponibil în cartea de eşantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Rolul HITL în Atenuarea Riscurilor AI

Acest conţinut nu este disponibil în cartea de eşantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Progrese Tehnologice şi Perspective de Viitor

Provocări şi Limitări ale Sistemelor HITL

Gestionarea Inteligentă a Erorilor

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Abordări Tradiţionale de Gestionare a Erorilor

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Diagnosticarea Contextuală a Erorilor

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Cum Funcționează

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Ingineria Prompturilor pentru Diagnosticarea Contextuală a Erorilor

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Generarea Augmentată prin Recuperare pentru Diagnosticarea Contextuală a Erorilor

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Raportarea Inteligentă a Erorilor

Prevenirea Predictivă a Erorilor

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Cum Funcționează

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Recuperarea Inteligentă din Erori

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Cum Funcționează

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Comunicare Personalizată a Erorilor

Cum Funcţionează

Flux de Lucru Adaptiv pentru Gestionarea Erorilor

Cum funcţionează

Controlul Calităţii

Acest conţinut nu este disponibil în cartea de eşantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Eval

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Problema

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Soluţia

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Cum Funcţionează

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Exemplu

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Consideraţii

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Înțelegerea Referințelor Etalon

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Cum Funcționează Evaluările Fără Referință

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Măsură de Protecţie

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Problemă

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Soluţie

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Cum funcţionează

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Exemplu

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Consideraţii

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Măsuri de Protecţie şi Evaluări: Două Feţe ale Aceleiaşi Monede

Interschimbabilitatea Măsurilor de Protecţie şi a Evaluărilor fără Referinţă

Implementarea Măsurilor de Protecţie şi a Evaluărilor cu Dublu Scop

Glosar

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Glosar

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

A

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

B

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

C

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

D

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

E

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

F

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

G

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

H

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

I

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

J

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

K

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

L

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

M

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

N

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

O

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

P

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Q

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

R

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

S

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

T

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

U

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

V

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

W

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Z

Acest conținut nu este disponibil în cartea de eșantion. Cartea poate fi cumpărată pe Leanpub la http://leanpub.com/patterns-of-application-development-using-ai-ro.

Index