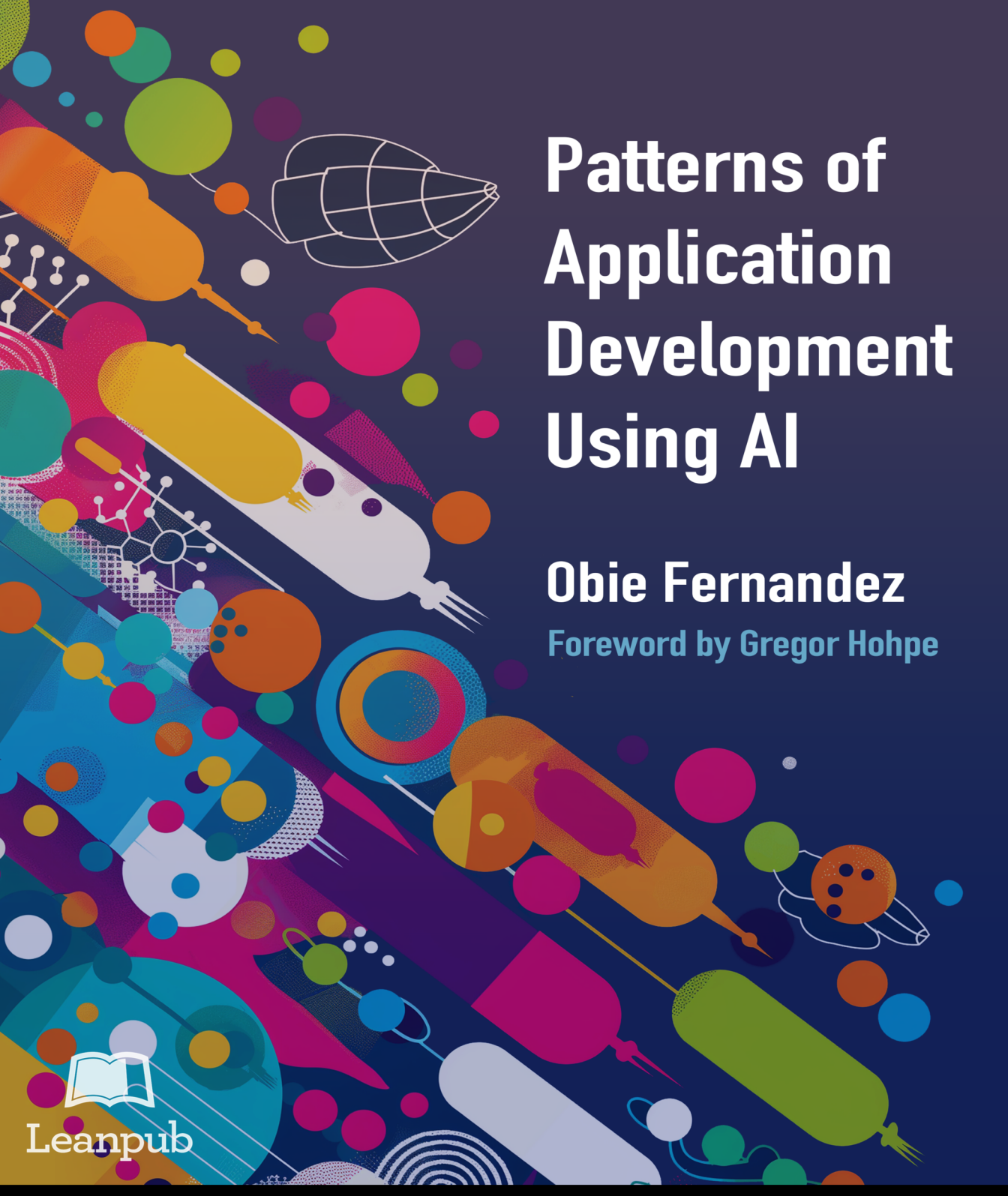


České Vydání

Vzory vývoje aplikací s využitím UI (České Vydání)

Obie Fernandez

Tato kniha se prodává na
http://leanpub.com/patterns-of-application-development-using-ai-cs

Tato verze byla publikována 2025-01-23

Mluvte o této knize na Twitteru!

Pomozte, prosím, Obie Fernandez šířením informace o této knize na Twitteru!

Navrhovaný hashtag pro tuto knihu je #poaduai.

Zjistěte, co ostatní lidé říkají o této knize. Stačí vyhledat tento hashtag na Twitteru:

#poaduai

Mé drsné královně, mé múze, mému světlu a lásce, Victorii

A také od Obie Fernandez

Patterns of Application Development Using AI

The Rails 8 Way

The Rails 7 Way

XML The Rails Way

Serverless

El Libro Principiante de Node

The Lean Enterprise

Obsah

Předmluva od Gregora Hohpe

Předmluva

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

O knize

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

O příkladech kódu

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Čemu se nevěnuji

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Pro koho je tato kniha určena

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Budování společného slovníku

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Jak se zapojit

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Poděkování

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Co je to s ilustracemi?

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

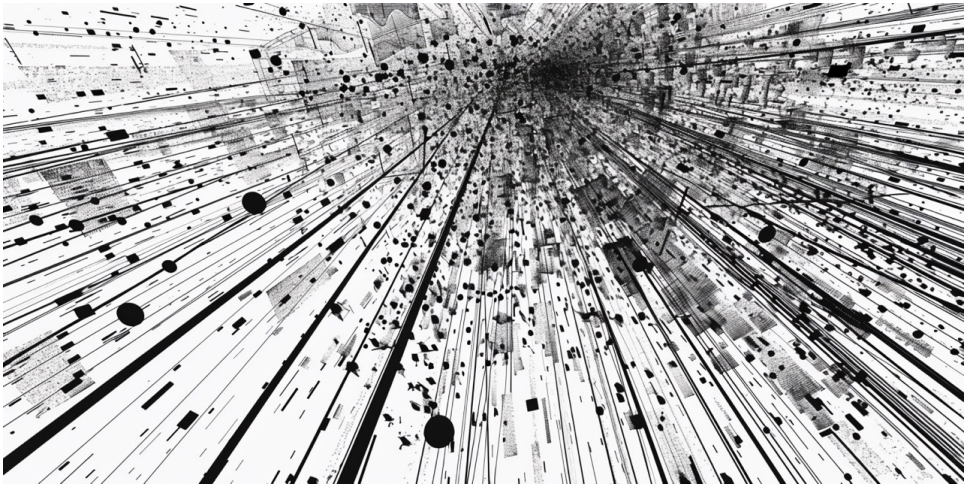
O Lean Publishingu

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

O autorovi

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Úvod



Pokud se těšíte na to, až začnete integrovat umělou inteligenci a velké jazykové modely (LLM) do svých programátorských projektů, můžete se směle pustit rovnou do vzorů a příkladů kódu uvedených v pozdějších kapitolách. Pro plné pochopení síly a potenciálu těchto vzorů však stojí za to věnovat chvíli porozumění širšímu kontextu a ucelenému přístupu, který představují.

Tyto vzory nejsou pouze sbírkou izolovaných technik, ale představují jednotný rámec pro integraci umělé inteligence do vašich aplikací. Já používám Ruby on Rails, ale tyto vzory by měly fungovat prakticky v jakémkoli jiném programovacím prostředí. Zabývají se širokou škálou aspektů, od správy dat a optimalizace výkonu až po uživatelskou zkušenost a bezpečnost, a poskytují komplexní sadu nástrojů pro vylepšení tradičních programovacích postupů pomocí možností umělé inteligence.

Každá kategorie vzorů řeší konkrétní výzvu nebo příležitost, která vzniká při začleňování komponent umělé inteligence do vaší aplikace. Pochopením vztahů a synergií mezi

těmito vzory můžete činit informovaná rozhodnutí o tom, kde a jak umělou inteligenci nejefektivněji využít.

Vzory nikdy nejsou předepisujícími řešeními a nemělo by se s nimi tak zacházet. Jsou zamýšleny jako přizpůsobitelné stavební bloky, které by měly být upraveny podle jedinečných požadavků a omezení vaší vlastní aplikace. Úspěšné použití těchto vzorů (stejně jako jakýchkoli jiných v oblasti softwaru) závisí na hlubokém porozumění problémové oblasti, potřebám uživatelů a celkové technické architektuře vašeho projektu.

Úvahy o softwarové architektuře

Začal jsem programovat v 80. letech a byl jsem součástí hackerské scény, přičemž jsem nikdy neztratil své hackerské smýšlení, ani poté, co jsem se stal profesionálním vývojářem softwaru. Od začátku jsem vždy měl zdravou skepsi ohledně toho, jakou hodnotu vlastně přinášejí softwaroví architekti ze svých slonovinových věží.

Jedním z důvodů, proč jsem osobně tak nadšený ze změn, které přináší tato mocná nová vlna technologie umělé inteligence, je její dopad na to, co považujeme za rozhodnutí *softwarové architektury*. Zpochybňuje tradiční představy o tom, co představuje "správný" způsob návrhu a implementace našich softwarových projektů. Také zpochybňuje, zda lze architekturu stále považovat především za *ty části systému, které je těžké změnit*, protože vylepšení pomocí umělé inteligence usnadňuje změnu jakékoli části vašeho projektu kdykoli více než kdy předtím.

Možná vstupujeme do vrcholných let "postmoderního" přístupu k softwarovému inženýrství. V tomto kontextu postmoderní označuje zásadní odklon od tradičních paradigmat, kde byli vývojáři odpovědní za napsání a údržbu každého řádku kódu. Místo toho přijímá myšlenku delegování úkolů, jako je manipulace s daty, komplexní algoritmy a dokonce celé části aplikační logiky, na knihovny třetích stran a externí API. Tento postmoderní posun představuje významný odklon od konvenční moudrosti

budování aplikací od základů a vyzývá vývojáře k přehodnocení jejich role v procesu vývoje.

Vždy jsem věřil, že dobří programátoři píší pouze kód, který je absolutně nezbytné napsat, na základě učení Larryho Walla a dalších hackerských osobností jako on. Minimalizací množství napsaného kódu se můžeme pohybovat rychleji, snížit prostor pro chyby, zjednodušit údržbu a zlepšit celkovou spolehlivost našich aplikací. Méně kódu nám umožňuje soustředit se na základní byznysovou logiku a uživatelskou zkušenost a delegovat ostatní práci na jiné služby.

Nyní, když systémy poháněné umělou inteligencí mohou zvládat úkoly, které byly dříve výhradní doménou kódu psaného člověkem, bychom měli být schopni být ještě produktivnější a agilnější, s větším zaměřením než kdy předtím na vytváření byznysové hodnoty a uživatelské zkušenosti.

Samozřejmě existují kompromisy při delegování obrovských částí vašeho projektu na systémy umělé inteligence, jako je potenciální ztráta kontroly a potřeba robustních monitorovacích a zpětnovazebních mechanismů. Proto to vyžaduje novou sadu dovedností a znalostí, včetně alespoň základního porozumění tomu, jak umělá inteligence funguje.

Co je velký jazykový model?

Velké jazykové modely (LLM) jsou typem umělé inteligence, který získal významnou pozornost v posledních letech, zejména od spuštění GPT-3 společností OpenAI v roce 2020. LLM jsou navrženy ke zpracování, porozumění a generování lidského jazyka s pozoruhodnou přesností a plynulostí. V této části se krátce podíváme na to, jak LLM fungují a proč jsou vhodné pro budování inteligentních systémových komponent.

V jádru jsou LLM založeny na algoritmech hlubokého učení, konkrétně na neuronových sítích. Tyto sítě se skládají z propojených uzlů neboli neuronů, které zpracovávají

a přenášejí informace. Architekturou volby pro LLM je často model Transformer, který se ukázal jako vysoce efektivní při zpracování sekvenčních dat, jako je text.

Transformerové modely jsou založeny na mechanismu pozornosti a používají se především pro úlohy zahrnující sekvenční data, jako je zpracování přirozeného jazyka. Transformery zpracovávají vstupní data najednou, nikoli sekvenčně, což jim umožňuje efektivněji zachytit dlouhodobé závislosti. Mají vrstvy mechanismů pozornosti, které modelu pomáhají soustředit se na různé části vstupních dat, aby pochopil kontext a vztahy.

Proces trénování velkých jazykových modelů zahrnuje vystavení modelu obrovskému množství textových dat, jako jsou knihy, články, webové stránky a repozitáře kódu. Během tréninku se model učí rozpoznávat vzory, vztahy a struktury v textu. Zachycuje statistické vlastnosti jazyka, jako jsou gramatická pravidla, slovní asociace a kontextuální významy.

Jednou z klíčových technik používaných při trénování velkých jazykových modelů je neřízené učení. To znamená, že se model učí z dat bez explicitního označování nebo vedení. Objevuje vzory a reprezentace samostatně analyzováním společného výskytu slov a frází v trénovacích datech. To umožňuje velkým jazykovým modelům vyvinout hluboké porozumění jazyku a jeho složitostem.

Dalším důležitým aspektem velkých jazykových modelů je jejich schopnost pracovat s *kontextem*. Při zpracování textu berou velké jazykové modely v úvahu nejen jednotlivá slova, ale také okolní kontext. Zohledňují předchozí slova, věty a dokonce i odstavce, aby pochopily význam a záměr textu. Toto kontextuální porozumění umožňuje velkým jazykovým modelům generovat koherentní a relevantní odpovědi. Jedním z hlavních způsobů, jak hodnotíme schopnosti daného jazykového modelu, je posouzení velikosti kontextu, který dokáže zvážit při generování odpovědí.

Po natrénování lze velké jazykové modely použít pro širokou škálu jazykových úloh. Dokážou generovat text podobný lidskému, odpovídat na otázky, shrnovat dokumenty, překládat jazyky a dokonce psát kód. Všestrannost velkých jazykových modelů je

činí cennými pro vytváření inteligentních systémových komponent, které mohou komunikovat s uživateli, zpracovávat a analyzovat textová data a generovat smysluplné výstupy.

Začleněním velkých jazykových modelů do architektury aplikací můžete vytvářet AI komponenty, které rozumí a zpracovávají uživatelské vstupy, generují dynamický obsah a poskytují inteligentní doporučení nebo akce. Práce s velkými jazykovými modely však vyžaduje pečlivé zvážení požadavků na zdroje a kompromisů ve výkonu. Velké jazykové modely jsou výpočetně náročné a mohou vyžadovat značný výpočetní výkon a paměť (jinými slovy, peníze) pro provoz. Většina z nás bude muset posoudit nákladové důsledky integrace velkých jazykových modelů do našich aplikací a podle toho jednat.

Porozumění inferenci

Inference označuje proces, kterým model generuje predikce nebo výstupy na základě nových, dosud neviděných dat. Je to fáze, kdy se natrénovaný model používá k rozhodování nebo generování textu, obrázků nebo jiného obsahu v reakci na uživatelské vstupy.

Během fáze trénování se AI model učí z velkého datasetu úpravou svých parametrů tak, aby minimalizoval chyby ve svých predikcích. Po natrénování může model aplikovat to, co se naučil, na nová data. Inference je způsob, jakým model využívá naučené vzory a znalosti k generování výstupů.

Pro velké jazykové modely inference zahrnuje přijetí promptu nebo vstupního textu a vytvoření koherentní a kontextuálně relevantní odpovědi jako proudu *tokenů* (o kterých budeme brzy mluvit). Může jít o odpověď na otázku, dokončení věty, generování příběhu nebo překlad textu, mezi mnoha dalšími úlohami.

Na rozdíl od způsobu, jakým přemýšlíme my, "myšlení" AI modelu prostřednictvím inference probíhá v jedné bezstavové operaci. To znamená, že jeho myšlení je omezeno na proces generování. Doslova musí myslet nahlas, jako kdybyste mi položili otázku a přijímali ode mě odpověď pouze ve stylu "proudu vědomí".

Velké jazykové modely přicházejí v mnoha velikostech a variantách

Zatímco prakticky všechny populární velké jazykové modely (LLM) jsou založeny na stejné základní transformerové architektuře a jsou trénovány na obrovských textových datasetech, přicházejí v různých velikostech a jsou doladěny pro různé účely. Velikost LLM, měřená počtem parametrů v jeho neuronové síti, má velký vliv na jeho schopnosti. Větší modely s více parametry, jako je GPT--4, o kterém se říká, že má 1 až 2 biliony parametrů, jsou obecně znalější a schopnější než menší modely. Větší modely však také vyžadují mnohem více výpočetního výkonu, což se promítá do vyšších nákladů při jejich používání prostřednictvím API volání.

Aby byly LLM praktičtější a přizpůsobené konkrétním případům použití, jsou základní modely často doladěny na více cílených datasetech. Například LLM může být trénován na velkém korpusu dialogů, aby se specializoval na konverzační AI. Jiné jsou trénovány na kódu, aby získaly programátorské znalosti. Existují dokonce modely, které jsou speciálně trénovány pro interakce s uživateli ve stylu hraní rolí!

Modely založené na vyhledávání vs. generativní modely

Ve světě velkých jazykových modelů (LLM) existují dva hlavní přístupy ke generování odpovědí: modely založené na vyhledávání a generativní modely. Každý přístup má své silné a slabé stránky a pochopení rozdílů mezi nimi vám může pomoci vybrat správný model pro váš konkrétní případ použití.

Modely založené na vyhledávání

Modely založené na vyhledávání, známé také jako modely pro získávání informací, generují odpovědi prohledáváním rozsáhlé databáze existujících textů a výběrem nejrelevantnějších pasáží na základě vstupního dotazu. Tyto modely nevytvářejí nový text od základu, ale spíše spojují úryvky z databáze do souvislé odpovědi.

Jednou z hlavních výhod modelů založených na vyhledávání je jejich schopnost poskytovat fakticky přesné a aktuální informace. Protože se spoléhají na databázi kurátovaných textů, mohou čerpat relevantní informace ze spolehlivých zdrojů a předkládat je uživateli. Díky tomu jsou vhodné pro aplikace, které vyžadují přesné, faktické odpovědi, jako jsou systémy pro zodpovídání otázek nebo znalostní báze.

Modely založené na vyhledávání mají však určitá omezení. Jsou pouze tak dobré jako databáze, kterou prohledávají, takže kvalita a pokrytí databáze přímo ovlivňují výkon modelu. Kromě toho mohou tyto modely mít potíže s generováním souvislých a přirozeně znějících odpovědí, protože jsou omezeny na text dostupný v databázi.

V této knize se nezabýváme používáním čistě vyhledávacích modelů.

Generativní modely

Generativní modely naopak vytvářejí nový text od základu na základě vzorců a vztahů, které se naučily během tréninku. Tyto modely využívají své porozumění jazyku k vytváření nových odpovědí, které jsou přizpůsobeny vstupnímu zadání.

Hlavní předností generativních modelů je jejich schopnost vytvářet kreativní, souvislý a kontextově relevantní text. Mohou vést otevřené konverzace, generovat příběhy a dokonce psát kód. Díky tomu jsou ideální pro aplikace, které vyžadují otevřenější a dynamičtější interakce, jako jsou chatboti, tvorba obsahu a asistenti pro kreativní psaní.

Generativní modely však mohou někdy produkovat nekonzistentní nebo fakticky nesprávné informace, protože se spoléhají spíše na vzorce naučené během tréninku

než na kurátovanou databázi faktů. Mohou být také náchylnější k předpojatosti a halucinacím, kdy generují text, který je věrohodný, ale ne nutně pravdivý.

Příklady generativních LLM zahrnují řadu GPT od OpenAI (GPT-3, GPT-4) a Claude od Anthropic.

Hybridní modely

Několik komerčně dostupných LLM kombinuje oba přístupy - vyhledávání i generování - v hybridním modelu. Tyto modely používají techniky vyhledávání k nalezení relevantních informací z databáze a poté používají generativní techniky k syntéze těchto informací do souvislé odpovědi.

Hybridní modely se snaží kombinovat faktickou přesnost modelů založených na vyhledávání s možnostmi generování přirozeného jazyka generativních modelů. Mohou poskytovat spolehlivější a aktuálnější informace při zachování schopnosti vést otevřené konverzace.

Při výběru mezi modely založenými na vyhledávání a generativními modely byste měli zvážit konkrétní požadavky vaší aplikace. Pokud je hlavním cílem poskytovat přesné, faktické informace, může být nejlepší volbou model založený na vyhledávání. Pokud aplikace vyžaduje otevřenější a kreativnější interakce, může být vhodnější generativní model. Hybridní modely nabízejí rovnováhu mezi oběma přístupy a mohou být dobrou volbou pro aplikace, které vyžadují jak faktickou přesnost, tak generování přirozeného jazyka.

V konečném důsledku závisí volba mezi modely založenými na vyhledávání a generativními modely na konkrétním případu použití a kompromisech mezi přesností, kreativitou a flexibilitou. Pochopením silných stránek a omezení každého přístupu můžete činit informovaná rozhodnutí.

Instrukční ladění

Instrukční ladění je podmnožina konverzačního tréninku, která používá člověkem napsané prompty a odpovědi k tomu, aby se LLM lépe naučily chápat a následovat instrukce. Zde jsou jednoduché příklady ilustrující, jak instrukční ladění na člověkem napsaných promptech a odpovědích zlepšuje schopnost LLM chápat a následovat instrukce:

Příklad 1: Zákaznická podpora

Instrukce: "Poskytněte odpověď zákazníkovi, který žádá o vrácení peněz."

Člověkem napsaný prompt: "Rád bych požádal o vrácení peněz za svůj nedávný nákup. Produkt nesplnil moje očekávání."

Člověkem napsaná odpověď: "Je mi líto, že produkt nesplnil vaše očekávání. Pro zpracování vrácení peněz prosím uveďte číslo vaší objednávky a důvod vrácení. Vaši žádost prověříme a ozveme se vám do 2-3 pracovních dnů."

Příklad 2: Vzdělávací asistence

Instrukce: "Vysvětlete koncept fotosyntézy desetiletému dítěti."

Lidmi psaný dotaz: "Můžeš vysvětlit, jak si rostliny vyrábějí potravu?"

Lidmi psaná odpověď: "Jistě! Rostliny si vyrábějí potravu procesem zvaným fotosyntéza. Využívají sluneční světlo, vodu a oxid uhličitý k vytvoření potravy a kyslíku. Listy rostliny obsahují speciální část zvanou chlorofyl, která jim pomáhá zachytávat sluneční světlo."

Příklad 3: Osobní asistent

Instrukce: "Naplánuj schůzku na příští úterý ve 15:00."

> Lidmi psaný dotaz: “Můžeš mi naplánovat schůzku na úterý ve tři odpoledne?”
>
> Lidmi psaná odpověď: “Jistě, naplánoval jsem vám schůzku na příští úterý v 15:00. Potřebujete ještě něco?”

Výsledkem je různorodý ekosystém velkých jazykových modelů různých velikostí a s různými specializacemi. Menší modely v rozsahu 1-7 miliard parametrů poskytují dobré obecné jazykové schopnosti a jsou přitom efektivnější pro provoz.

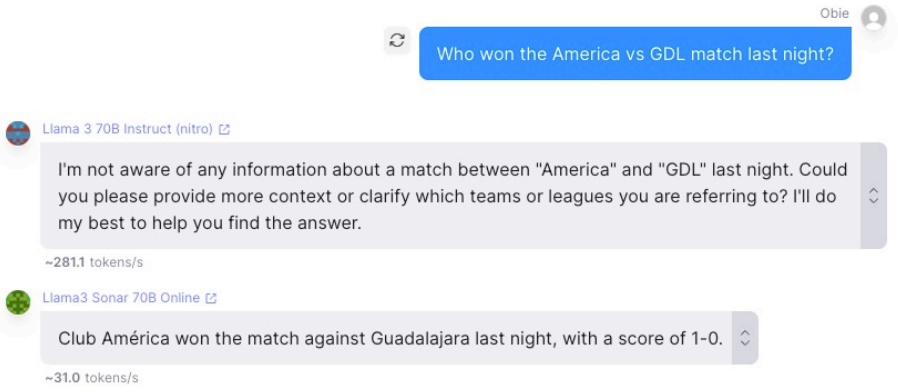
- Mistral 7B
- Llama 3 8B
- Gemma 7B

Středně velké modely s přibližně 30-70 miliardami parametrů nabízejí silnější schopnosti uvažování a následování instrukcí.

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

Při výběru velkého jazykového modelu pro začlenění do aplikace musíte vyvážit schopnosti modelu s praktickými faktory jako jsou náklady, latence, délka kontextu a filtrování obsahu. Menší modely doladěné na instrukce jsou často nejlepší volbou pro jednodušší jazykové úlohy, zatímco největší modely mohou být potřebné pro složité uvažování nebo analýzu. Důležitým faktorem je také trénovací dataset modelu, který určuje datum ukončení znalostí modelu.

Některé modely, jako například některé od Perplexity, jsou připojeny ke zdrojům informací v reálném čase, takže v podstatě nemají žádné datum ukončení znalostí. Když jim položíte otázky, dokážou samostatně rozhodnout o provedení webového vyhledávání a načtení libovolných webových stránek pro generování odpovědi.

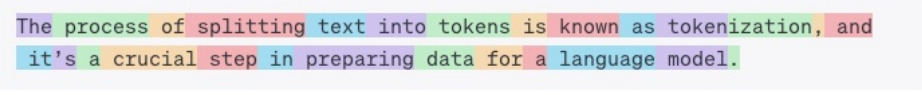


obrázkem 1. Llama3 s online přístupem a bez něj

V konečném důsledku neexistuje univerzální velký jazykový model. Pro výběr správného modelu pro daný případ použití je klíčové porozumění rozdílům ve velikosti modelu, architektuře a tréninku. Experimentování s různými modely je jediný praktický způsob, jak zjistit, které z nich poskytují nejlepší výkon pro danou úlohu.

Tokenizace: Rozdělování textu na části

Než může velký jazykový model zpracovat text, musí být tento text rozdělen na menší jednotky zvané *tokeny*. Tokeny mohou být jednotlivá slova, části slov nebo dokonce jednotlivé znaky. Proces rozdělování textu na tokeny se nazývá tokenizace a je to klíčový krok v přípravě dat pro jazykový model.



obrázkem 2. Tato věta obsahuje 27 tokenů

Různé velké jazykové modely používají různé strategie tokenizace, což může mít významný vliv na výkon a schopnosti modelu. Mezi běžné tokenizéry používané

velkými jazykovými modely patří:

- **GPT (Kódování párů bajtů)**: GPT tokenizéry používají techniku zvanou kódování párů bajtů (BPE) k rozdělení textu na podslovovné jednotky. BPE iterativně spojuje nejčastější páry bajtů v textovém korpusu a vytváří tak slovník podslovovných tokenů. To umožňuje tokenizéru zpracovat vzácná a nová slova jejich rozdělením na běžnější podslovovné části. GPT tokenizéry jsou používány modely jako GPT--3 a GPT--4.

- **Llama (SentencePiece)**: Tokenizátory Llama používají knihovnu SentencePiece, což je neřízený textový tokenizátor a detokenizátor. SentencePiece pracuje se vstupním textem jako se sekvencí znaků Unicode a učí se slovník podslov na základě trénovacího korpusu. Dokáže zpracovat jakýkoliv jazyk, který lze zakódovat v Unicode, což ho činí vhodným pro vícejazyčné modely. Tokenizátory Llama jsou používány modely jako Meta's Llama a Alpaca.

- **SentencePiece (Unigram)**: Tokenizátory SentencePiece mohou také používat jiný algoritmus nazvaný Unigram, který je založen na technice regularizace podslov. Tokenizace Unigram určuje optimální slovník podslov na základě unigramového jazykového modelu, který přiřazuje pravděpodobnosti jednotlivým podslovním jednotkám. Tento přístup může produkovat sémanticky významnější podslova ve srovnání s BPE. SentencePiece s Unigramem používají modely jako Google T5 a BERT.

- **Google Gemini (Multimodální tokenizace)**: Google Gemini používá tokenizační schéma navržené pro zpracování různých typů dat, včetně textu, obrázků, zvuku, videí a kódu. Tato multimodální schopnost umožňuje Gemini zpracovávat a integrovat různé formy informací. Je pozoruhodné, že Google Gemini 1.5 Pro má kontextové okno, které zvládne miliony tokenů, což je mnohem více než předchozí modely. Toto rozsáhlé kontextové okno umožňuje modelu zpracovávat

větší kontext, což potenciálně vede k přesnějším odpovědím. Je však důležité poznamenat, že tokenizační schéma Gemini je mnohem blíže jednomu tokenu na znak než u jiných modelů. To znamená, že skutečné náklady na používání modelů Gemini mohou být výrazně vyšší než očekávané, pokud jste zvyklí používat modely jako GPT, protože Google účtuje ceny na základě znaků spíše než tokenů.

Volba tokenizátoru ovlivňuje několik aspektů LLM, včetně:

- **Velikost slovníku**: Tokenizátor určuje velikost slovníku modelu, což je sada unikátních tokenů, které rozpoznává. Větší a podrobnější slovník může modelu pomoci zpracovat širší škálu slov a frází a dokonce se stát multimodálním (schopným porozumět a generovat více než jen text), ale také zvyšuje paměťové nároky modelu a výpočetní složitost.
- **Zpracování vzácných a neznámých slov**: Tokenizátory, které používají podslovní jednotky, jako BPE a SentencePiece, mohou rozdělit vzácná a neznámá slova na běžnější části podslov. To umožňuje modelu dělat kvalifikované odhady o významu slov, která předtím neviděl, na základě podslov, která obsahují.
- **Vícejazyčná podpora**: Tokenizátory jako SentencePiece, které dokážou zpracovat jakýkoliv jazyk kódovatelný v Unicode, jsou dobře uzpůsobené pro vícejazyčné modely, které potřebují zpracovávat text v různých jazycích.

Při výběru LLM pro konkrétní aplikaci je důležité zvážit tokenizátor, který používá, a jak dobře odpovídá specifickým potřebám zpracování jazyka pro daný úkol. Tokenizátor může mít významný vliv na schopnost modelu zpracovávat oborovou terminologii, vzácná slova a vícejazyčný text.

Velikost kontextu: Kolik informací může jazykový model využít během inference?

Při diskusi o jazykových modelech se velikostí kontextu rozumí množství textu, které model může zvážit při zpracování nebo generování svých odpovědí. Je to v podstatě

míra toho, kolik informací si model dokáže "zapamatovat" a použít pro své výstupy (vyjádřeno v tokenech). Velikost kontextu jazykového modelu může mít významný dopad na jeho schopnosti a typy úloh, které může efektivně provádět.

Co je velikost kontextu?

Technicky vzato je velikost kontextu určena počtem tokenů (slov nebo částí slov), které jazykový model může zpracovat v jedné vstupní sekvenci. Často se tomu říká "rozsah pozornosti" nebo "kontextové okno" modelu. Čím větší je velikost kontextu, tím více textu může model najednou zvážit při generování odpovědi nebo provádění úkolu.

Různé jazykové modely mají různé velikosti kontextu, od několika set tokenů až po miliony tokenů. Pro představu, typický odstavec textu může obsahovat přibližně 100-150 tokenů, zatímco celá kniha může obsahovat desítky či stovky tisíc tokenů.

> Existují dokonce práce na efektivních metodách pro škálování Transformer-based Large Language Models (LLMs) na nekonečně dlouhé vstupy s omezenou pamětí a výpočetní náročností.

Proč je velikost kontextu důležitá?

Velikost kontextu jazykového modelu má významný vliv na jeho schopnost porozumět a generovat souvislý, kontextově relevantní text. Zde jsou některé klíčové důvody, proč na velikosti kontextu záleží:

1. **Porozumění dlouhým textům**: Modely s větší velikostí kontextu dokáží lépe pochopit a analyzovat delší texty, jako jsou články, zprávy nebo dokonce celé knihy. To je zásadní pro úlohy jako sumarizace dokumentů, zodpovídání otázek a analýza obsahu.

2. **Udržení koherence**: Větší kontextové okno umožňuje modelu udržet koherenci a konzistenci v delších úsecích výstupu. To je důležité pro úlohy jako generování příběhů, dialogové systémy a tvorba obsahu, kde je zásadní udržet konzistentní vyprávění nebo téma. Je to také naprosto klíčové při používání LLM pro generování nebo transformaci strukturovaných dat.
3. **Zachycení závislostí na dlouhou vzdálenost**: Některé jazykové úlohy vyžadují porozumění vztahům mezi slovy nebo frázemi, které jsou v textu od sebe vzdálené. Modely s větší velikostí kontextu jsou lépe vybaveny pro zachycení těchto vzdálených závislostí, což může být důležité pro úlohy jako analýza sentimentu, překlad a porozumění jazyku.
4. **Zvládání složitých instrukcí**: V aplikacích, kde se jazykové modely používají k následování složitých, vícekrokových instrukcí, větší velikost kontextu umožňuje modelu vzít v úvahu celou sadu instrukcí při generování odpovědi, místo jen několika posledních slov.

Příklady jazykových modelů s různými velikostmi kontextu

Zde je několik příkladů jazykových modelů s různými velikostmi kontextu:

- OpenAI GPT-3.5 Turbo: 4 095 tokenů
- Mistral 7B Instruct: 32 768 tokenů
- Anthropic Claude v1: 100 000 tokenů
- OpenAI GPT-4 Turbo: 128 000 tokenů
- Anthropic Claude v2: 200 000 tokenů
- Google Gemini Pro 1.5: 2,8M tokenů

Jak můžete vidět, mezi těmito modely je široký rozsah velikostí kontextu, od přibližně 4 000 tokenů u modelu OpenAI GPT-3.5 Turbo až po 200 000 tokenů u modelu Anthropic Claude v2. Některé modely, jako Google PaLM 2 a OpenAI GPT--4, nabízejí různé varianty s většími velikostmi kontextu (např. verze “32k”), které zvládnou ještě delší

vstupní sekvence. A v současnosti (duben 2024) se Google Gemini Pro chlubí téměř 3 miliony tokenů!

Je třeba poznamenat, že velikost kontextu se může lišit v závislosti na konkrétní implementaci a verzi daného modelu. Například původní model OpenAI GPT-4 má velikost kontextu 8 191 tokenů, zatímco pozdější varianty GPT-4, jako jsou Turbo a 4o, mají mnohem větší velikost kontextu 128 000 tokenů.

> Sam Altman přirovnal současná kontextová omezení ke kilobajtům pracovní paměti, se kterými se museli programátoři osobních počítačů potýkat v 80. letech, a řekl, že v blízké budoucnosti budeme schopni vměstnat "všechna vaše osobní data" do kontextu velkého jazykového modelu.

Výběr správné velikosti kontextu

Při výběru jazykového modelu pro konkrétní aplikaci je důležité zvážit požadavky na velikost kontextu pro danou úlohu. Pro úlohy zahrnující krátké, izolované texty, jako je analýza sentimentu nebo jednoduché zodpovídání otázek, může být dostačující menší velikost kontextu. Pro úlohy vyžadující porozumění a generování delších, složitějších textů bude pravděpodobně nutná větší velikost kontextu.

Je třeba poznamenat, že větší velikosti kontextu často přinášejí zvýšené výpočetní náklady a pomalejší zpracování, protože model musí při generování odpovědi zvažovat více informací. Proto musíte při výběru jazykového modelu pro vaši aplikaci najít rovnováhu mezi velikostí kontextu a výkonem.

> Proč tedy jednoduše nevybrat model s největší velikostí kontextu a nenaplnit ho co největším množstvím informací? No, kromě výkonnostních faktorů je hlavním důvodem cena. V březnu 2024 vás *jediný* cyklus dotaz-odpověď pomocí Google Gemini Pro 1.5 s plným kontextem bude stát téměř 8 dolarů (USD). Pokud máte případ použití, který tuto cenu ospravedlňuje, tím lépe! Ale pro většinu aplikací je to prostě o řády příliš drahé.

Hledání jehel v kupkách sena

Koncept hledání jehly v kupce sena je dlouho používanou metaforou pro výzvy spojené s vyhledáváním ve velkých datových souborech. V oblasti velkých jazykových modelů (LLM) tuto analogii mírně upravujeme. Představte si, že nehledáme jen jeden fakt ukrytý v rozsáhlém textu (jako je třeba kompletní antologie esejů Paula Grahama), ale několik faktů rozptýlených po celém textu. Tento scénář se více podobá hledání několika jehel v rozlehlém poli, nikoliv jen v jedné kupce sena. A zde je háček: nejen že musíme tyto jehly najít, ale musíme je také propojit do souvislého celku.

Když mají LLM za úkol vyhledávat a uvažovat o více faktech zasazených do dlouhých kontextů, čelí dvojí výzvě. Zaprvé je tu přímočarý problém přesnosti vyhledávání—ta přirozeně klesá s rostoucím počtem faktů. To je očekávatelné; koneckonců, udržet přehled o více detailech napříč rozsáhlým textem je náročné i pro ty nejsofistikovanější modely.

Zadruhé, a možná ještě kritičtější, je výzva uvažování s těmito fakty. Jedna věc je fakta vysbírat; něco zcela jiného je syntetizovat je do souvislého vyprávění nebo odpovědi. Zde přichází skutečná zkouška. Výkon LLM v úlohách vyžadujících uvažování má tendenci degradovat více než u jednoduchých úloh vyhledávání. Tato degradace není jen otázkou objemu; jde o složitý tanec kontextu, relevance a vyvozování závěrů.

Proč k tomu dochází? Uvažujme o dynamice paměti a pozornosti v lidském poznávání,

která se do určité míry odráží i v LLM. Při zpracování velkého množství informací mohou LLM, podobně jako lidé, ztratit přehled o dřívějších detailech, když vstřebávají nové. To platí zejména u modelů, které nejsou explicitně navrženy tak, aby automaticky upřednostňovaly nebo se vracely k dřívějším částem textu.

Navíc, schopnost LLM propojit tyto získané fakty do souvislé odpovědi se podobá vytváření narativu. To vyžaduje nejen vyhledání informací, ale i hluboké porozumění a kontextuální umístění, což zůstává pro současnou umělou inteligenci značnou výzvou.

Co to tedy znamená pro nás jako vývojáře a integrátory těchto technologií? Musíme si být ostře vědomi těchto omezení při navrhování systémů, které spoléhají na LLM pro zpracování komplexních, dlouhých úloh. Pochopení toho, že výkon se může za určitých podmínek zhoršit, nám pomáhá nastavit realistická očekávání a vytvářet lepší záložní mechanismy nebo doplňkové strategie.

Modality: Za hranicemi textu

Zatímco většina dnešních jazykových modelů se zaměřuje na zpracování a generování textu, roste trend směrem k multimodálním modelům, které dokáží přirozeně přijímat a vytvářet více typů dat, jako jsou obrázky, zvuk a video. Tyto multimodální modely otevírají nové možnosti pro aplikace založené na umělé inteligenci, které dokáží porozumět a generovat obsah napříč různými modalitami.

Co jsou modality?

V kontextu jazykových modelů se modalitami rozumí různé typy dat, které model dokáže zpracovávat a generovat. Nejběžnější modalitou je text, který zahrnuje psaný jazyk v různých formách jako knihy, články, webové stránky a příspěvky na sociálních sítích. Existuje však několik dalších modalit, které jsou stále častěji začleňovány do jazykových modelů:

- **Obrázky**: Vizuální data jako fotografie, ilustrace a diagramy.

- **Audio**: Zvuková data jako řeč, hudba a zvuky prostředí.
- **Video**: Pohyblivá vizuální data, často doprovázená zvukem, jako jsou videoklipy a filmy.

Každá modalita představuje jedinečné výzvy a příležitosti pro jazykové modely. Například obrázky vyžadují, aby model porozuměl vizuálním konceptům a vztahům, zatímco audio vyžaduje, aby model zpracovával a generoval řeč a další zvuky.

Multimodální jazykové modely

Multimodální jazykové modely jsou navrženy tak, aby zvládaly více modalit v rámci jediného modelu. Tyto modely typicky obsahují specializované komponenty nebo vrstvy, které dokáží jak porozumět vstupům, tak generovat výstupní data v různých modalitách. Mezi významné příklady multimodálních jazykových modelů patří:

- **OpenAI GPT-4o**: GPT-4o je velký jazykový model, který přirozeně rozumí a zpracovává řečové audio kromě textu. Tato schopnost umožňuje GPT-4o provádět úkoly jako přepis mluveného jazyka, generování textu ze zvukových vstupů a poskytování odpovědí na základě mluvených dotazů.
- **OpenAI GPT-4 s vizuálním vstupem**: GPT--4 je velký jazykový model, který dokáže zpracovávat jak text, tak obrázky. Když dostane obrázek jako vstup, GPT-4 dokáže analyzovat obsah obrázku a generovat text, který popisuje nebo reaguje na vizuální informace.
- **Google Gemini**: Gemini je multimodální model, který zvládá text, obrázky a video. Používá jednotnou architekturu, která umožňuje křížové porozumění a generování mezi modalitami, což umožňuje úlohy jako popisování obrázků, sumarizace videa a vizuální zodpovídání otázek.
- **DALL-E a Stable Diffusion**: Přestože nejde o jazykové modely v tradičním smyslu, tyto modely demonstrují sílu multimodální umělé inteligence generováním obrázků z textových popisů. Ukazují potenciál modelů, které dokáží překládat mezi různými modalitami.

Výhody a aplikace multimodálních modelů

Multimodální jazykové modely nabízejí několik výhod a umožňují širokou škálu aplikací, včetně:

- **Vylepšené porozumění**: Zpracováním informací z více modalit mohou tyto modely získat komplexnější porozumění světu, podobně jako se lidé učí z různých smyslových vstupů.
- **Křížově modální generování**: Multimodální modely dokáží generovat obsah v jedné modalitě na základě vstupu z jiné modality, například vytvořit obrázek z textového popisu nebo generovat video shrnutí z psaného článku.
- **Přístupnost**: Multimodální modely mohou zpřístupnit informace překladem mezi modalitami, například generováním textových popisů obrázků pro zrakově postižené uživatele nebo vytvářením zvukových verzí psaného obsahu.
- **Kreativní aplikace**: Multimodální modely lze využít pro kreativní úkoly jako generování umění, hudby nebo videí na základě textových promptů, což otevírá nové možnosti pro umělce a tvůrce obsahu.

S pokračujícím vývojem multimodálních jazykových modelů budou pravděpodobně hrát stále důležitější roli ve vývoji aplikací založených na umělé inteligenci, které dokáží porozumět a generovat obsah napříč různými modalitami. To umožní přirozenější a intuitivnější interakce mezi lidmi a systémy umělé inteligence a také otevře nové možnosti pro kreativní vyjádření a šíření znalostí.

Ekosystémy poskytovatelů

Pokud jde o začlenění velkých jazykových modelů (LLM) do aplikací, máte na výběr z rostoucí řady možností. Každý hlavní poskytovatel LLM, jako je OpenAI, Anthropic, Google a Cohere, nabízí vlastní ekosystém modelů, API a nástrojů. Výběr správného poskytovatele zahrnuje zvážení různých faktorů, včetně ceny, výkonu, filtrování obsahu, ochrany dat a možností přizpůsobení.

OpenAI

OpenAI je jedním z nejznámějších poskytovatelů LLM, přičemž jeho série GPT (GPT-3, GPT-4) je široce využívána v různých aplikacích. OpenAI nabízí uživatelsky přívětivé API, které vám umožňuje snadno integrovat jejich modely do aplikací. Poskytují řadu modelů s různými schopnostmi a cenovými úrovněmi, od základního modelu Ada až po výkonný model Davinci.

Ekosystém OpenAI také zahrnuje nástroje jako OpenAI Playground, který vám umožňuje experimentovat s prompty a jemně dolaďovat modely pro konkrétní případy použití. Nabízejí možnosti filtrování obsahu, které pomáhají předcházet generování nevhodného nebo škodlivého obsahu.

Při přímém používání modelů OpenAI spoléhám na knihovnu ruby-openai od Alexe Rudalla.

Anthropic

Anthropic je další významný hráč v oblasti LLM, jehož modely Claude získávají popularitu díky silnému výkonu a etickým aspektům. Anthropic se zaměřuje na vývoj bezpečných a odpovědných systémů umělé inteligence s velkým důrazem na filtrování obsahu a vyhýbání se škodlivým výstupům.

Ekosystém Anthropic zahrnuje API Claude, které vám umožňuje integrovat model do jejich aplikací, a také nástroje pro prompt engineering a jemné doladění. Nabízejí také model Claude Instant, který zahrnuje možnosti webového vyhledávání pro aktuálnější a fakticky přesnější odpovědi.

Při přímém používání modelů Anthropic spoléhám na knihovnu anthrophic od Alexe Rudalla.

Google

Google vyvinul několik výkonných LLM, včetně modelů Gemini, BERT, T5 a PaLM. Tyto modely jsou známé svým silným výkonem v širokém spektru úloh zpracování přirozeného jazyka. Ekosystém Google zahrnuje knihovny TensorFlow a Keras, které poskytují nástroje a frameworky pro vytváření a trénování modelů strojového učení.

Google také nabízí Cloud AI Platform, která vám umožňuje snadno nasazovat a škálovat jejich modely v cloudu. Poskytují řadu předtrénovaných modelů a API pro úlohy jako analýza sentimentu, rozpoznávání entit a překlad.

Meta

Meta, dříve známá jako Facebook, významně investuje do vývoje velkých jazykových modelů, což dokládá vydání modelů jako LLaMA a OPT. Tyto modely vynikají svým silným výkonem v různých jazykových úlohách a jsou dostupné převážně prostřednictvím open-source kanálů, podporující závazek Mety k výzkumu a komunitní spolupráci.

Ekosystém Mety je primárně postavený kolem PyTorch, open-source knihovny pro strojové učení, která je oblíbená pro své dynamické výpočetní schopnosti a flexibilitu, usnadňující inovativní výzkum a vývoj umělé inteligence.

Kromě svých technických řešení klade Meta velký důraz na etický vývoj umělé inteligence. Implementuje robustní filtrování obsahu a zaměřuje se na snižování předpojatosti, což je v souladu s jejich širšími cíli bezpečnosti a odpovědnosti v aplikacích umělé inteligence.

Cohere

Cohere je novější účastník v oblasti LLM, který se zaměřuje na zpřístupnění a zjednodušení používání LLM oproti konkurenci. Jejich ekosystém zahrnuje Cohere

API, které poskytuje přístup k řadě předtrénovaných modelů pro úlohy jako generování textu, klasifikace a sumarizace.

Cohere také nabízí nástroje pro inženýrství promptů, dolaďování a filtrování obsahu. Zdůrazňují ochranu a bezpečnost dat s funkcemi jako šifrované úložiště dat a kontrola přístupu.

Ollama

Ollama je lokálně hostovaná platforma, která uživatelům umožňuje spravovat a nasazovat různé velké jazykové modely (LLM) lokálně na jejich počítačích, což jim dává úplnou kontrolu nad jejich AI modely bez závislosti na externích cloudových službách. Toto nastavení je ideální pro ty, kteří upřednostňují ochranu dat a chtějí provozovat své AI operace interně.

Platforma podporuje řadu modelů, včetně verzí Llama, Phi, Gemma a Mistral, které se liší velikostí a výpočetními požadavky. Ollama usnadňuje stahování a spouštění těchto modelů přímo z příkazového řádku pomocí jednoduchých příkazů jako `ollama run <model_name>` a je navržena pro práci na různých operačních systémech včetně macOS, Linux a Windows.

Pro vývojáře, kteří chtějí integrovat open-source modely do svých aplikací bez použití vzdáleného API, nabízí Ollama CLI pro správu životního cyklu modelů podobně jako nástroje pro správu kontejnerů. Podporuje také vlastní konfigurace a prompty, což umožňuje vysokou míru přizpůsobení pro specifické potřeby nebo případy použití.

Ollama je obzvláště vhodná pro technicky zdatné uživatele a vývojáře díky svému rozhraní příkazového řádku a flexibilitě, kterou nabízí při správě a nasazování AI modelů. Díky tomu je silným nástrojem pro firmy a jednotlivce, kteří potřebují robustní AI schopnosti bez kompromisů v oblasti bezpečnosti a kontroly.

Multi-modelové platformy

Kromě toho existují poskytovatelé, kteří hostují širokou škálu open-source modelů, jako jsou Together.ai a Groq.. Tyto platformy nabízejí flexibilitu a přizpůsobení, což vám umožňuje spouštět a v některých případech dokonce dolaďovat open-source modely podle vašich specifických potřeb. Například Together.ai poskytuje přístup k řadě open-source LLM, což uživatelům umožňuje experimentovat s různými modely a konfiguracemi. Groq se zaměřuje na poskytování ultra výkonného dokončování, které se v době psaní této knihy zdá být téměř magické.

Výběr poskytovatele LLM

Při výběru poskytovatele LLM byste měli zvážit faktory jako:

- **Ceny**: Různí poskytovatelé nabízejí různé cenové modely, od platby za použití až po předplatitelské plány. Při výběru poskytovatele je důležité zvážit očekávané využití a rozpočet.
- **Výkon**: Výkon LLM se může mezi poskytovateli výrazně lišit, proto je důležité před rozhodnutím otestovat modely na konkrétních případech použití.
- **Filtrování obsahu**: V závislosti na aplikaci může být filtrování obsahu kritickým faktorem. Někteří poskytovatelé nabízejí robustnější možnosti filtrování obsahu než jiní.
- **Ochrana dat**: Pokud aplikace pracuje s citlivými uživatelskými daty, je důležité vybrat poskytovatele se silnými postupy ochrany a bezpečnosti dat.
- **Přizpůsobení**: Někteří poskytovatelé nabízejí větší flexibilitu v oblasti dolaďování a přizpůsobování modelů pro specifické případy použití.

Konečný výběr poskytovatele LLM závisí na specifických požadavcích a omezeních aplikace. Pečlivým vyhodnocením možností a zvážením faktorů jako ceny, výkon a ochrana dat můžete vybrat poskytovatele, který nejlépe vyhovuje vašim potřebám.

Stojí také za zmínku, že prostředí LLM se neustále vyvíjí a pravidelně se objevují noví poskytovatelé a modely. Měli byste sledovat nejnovější vývoj a být otevřeni zkoumání nových možností, jak se objevují.

OpenRouter

V této knize budu výhradně používat OpenRouter jako mého preferovaného poskytovatele API. Důvod je jednoduchý: je to komplexní řešení pro všechny nejpopulárnější komerční a open-source modely. Pokud se nemůžete dočkat, až si vyzkoušíte nějaké AI kódování, jedním z nejlepších míst, kde začít, je moje vlastní OpenRouter Ruby Library.

Zamyšlení nad výkonem

Při začleňování jazykových modelů do aplikací je výkon klíčovým faktorem. Výkon jazykového modelu lze měřit z hlediska jeho *latence* (doba potřebná k vygenerování odpovědi) a *propustnosti* (počet požadavků, které může zpracovat za jednotku času).

Time to First Token (TTFT) je další důležitou metrikou výkonu, která je obzvláště relevantní pro chatboty a aplikace vyžadující interaktivní odpovědi v reálném čase. TTFT měří latenci od okamžiku přijetí požadavku uživatele do okamžiku vygenerování prvního slova (nebo tokenu) odpovědi. Tato metrika je zásadní pro zachování plynulého a poutavého uživatelského zážitku, protože zpožděné odpovědi mohou vést k frustraci uživatelů a jejich odrazení.

Tyto metriky výkonu mohou mít významný dopad na uživatelský zážitek a škálovatelnost aplikace.

Výkon jazykového modelu může ovlivnit několik faktorů, včetně:

Počet parametrů: Větší modely s více parametry obecně vyžadují více výpočetních zdrojů a mohou mít vyšší latenci a nižší propustnost ve srovnání s menšími modely.

Hardware: Výkon jazykového modelu se může výrazně lišit v závislosti na hardwaru, na kterém běží. Poskytovatelé cloudových služeb nabízejí instance GPU a TPU optimalizované pro strojové učení, které mohou výrazně urychlit inferenci modelu.

Jednou z příjemných věcí na OpenRouteru je, že u mnoha nabízených modelů máte na výběr z různých poskytovatelů cloudových služeb s různými výkonnostními profily a náklady.

Kvantizace: Kvantizační techniky lze použít ke snížení paměťové náročnosti a výpočetních požadavků modelu reprezentací vah a aktivací pomocí datových typů s nižší přesností. To může zlepšit výkon bez významného obětování kvality. Jako vývojář aplikací se pravděpodobně nebudete zabývat trénováním vlastních modelů s různými úrovněmi kvantizace, ale je dobré být alespoň obeznámen s terminologií.

Dávkové zpracování: Zpracování více požadavků současně v dávkách může zlepšit propustnost díky amortizaci režie načítání modelu a přenosu dat.

Kešování: Ukládání výsledků často používaných promptů nebo vstupních sekvencí do mezipaměti může snížit počet inferenčních požadavků a zlepšit celkový výkon.

Při výběru jazykového modelu pro produkční aplikaci je důležité otestovat jeho výkon na reprezentativních pracovních zátěžích a hardwarových konfiguracích. To může pomoci identifikovat potenciální úzká místa a zajistit, že model splní požadované výkonnostní cíle.

Stojí také za to zvážit kompromisy mezi výkonem modelu a dalšími faktory, jako jsou náklady, flexibilita a snadnost integrace. Například použití menšího, méně nákladného modelu s nižší latencí může být vhodnější pro aplikace vyžadující odpovědi v reálném čase, zatímco větší, výkonnější model může být vhodnější pro dávkové zpracování nebo úlohy komplexního uvažování.

Experimenty s různými modely LLM

Volba LLM je zřídkakdy trvalým rozhodnutím. Vzhledem k tomu, že jsou pravidelně vydávány nové a vylepšené modely, je dobré budovat aplikace modulárním způsobem, který umožňuje v průběhu času zaměňovat různé jazykové modely. Prompty a datasety lze často používat napříč modely s minimálními změnami. To vám umožňuje využívat nejnovější pokroky v jazykovém modelování, aniž byste museli zcela přepracovávat své aplikace.

Možnost snadno přepínat mezi širokou škálou modelů je dalším důvodem, proč mám OpenRouter tak rád.

Při přechodu na nový jazykový model je důležité důkladně otestovat a ověřit jeho výkon a kvalitu výstupu, abyste se ujistili, že splňuje požadavky aplikace. To může zahrnovat přetrénování nebo dolaďování modelu na doménově specifických datech a aktualizaci všech navazujících komponent, které závisí na výstupech modelu.

Navrhováním aplikací s ohledem na výkon a modularitu můžete vytvářet škálovatelné, efektivní a do budoucna připravené systémy, které se dokáží přizpůsobit rychle se vyvíjející oblasti technologie jazykového modelování.

Složené systémy umělé inteligence

Než uzavřeme náš úvod, stojí za zmínku, že před rokem 2023 a explozí zájmu o generativní AI, kterou vyvolal ChatGPT, se tradiční přístupy k AI obvykle spoléhaly na integraci jednotlivých, uzavřených modelů. Naproti tomu *složené systémy umělé inteligence* využívají komplexní řetězce propojených komponent, které spolupracují na dosažení inteligentního chování.

V jádru se složené systémy umělé inteligence skládají z více modulů, z nichž každý je navržen pro provádění specifických úkolů nebo funkcí. Tyto moduly mohou zahrnovat

generátory, vyhledávače, hodnotící systémy, klasifikátory a různé další specializované komponenty. Rozdělením celkového systému na menší, zaměřené jednotky mohou vývojáři vytvářet flexibilnější, škálovatelnější a udržovatelnější architektury AI.

Jednou z klíčových výhod složených systémů umělé inteligence je jejich schopnost kombinovat silné stránky různých technik a modelů AI. Například systém může využívat velký jazykový model (LLM) pro porozumění a generování přirozeného jazyka, zatímco používá samostatný model pro vyhledávání informací nebo rozhodování založené na pravidlech. Tento modulární přístup vám umožňuje vybrat nejlepší nástroje a techniky pro každý konkrétní úkol, místo spoléhání se na univerzální řešení.

Vytváření složených systémů umělé inteligence však přináší i jedinečné výzvy. Zejména zajištění celkové koherence a konzistence chování systému vyžaduje robustní testování, monitoring a řídicí mechanismy.

Příchod výkonných LLM jako GPT--4 nám umožňuje experimentovat se složenými systémy AI snadněji než kdy předtím, protože tyto pokročilé modely jsou schopné zastávat více rolí v rámci složeného systému, jako je klasifikace, řazení a generování, kromě jejich schopností porozumění přirozenému jazyku. Tato všestrannost umožňuje vývojářům rychle vytvářet prototypy a iterovat na architekturách složených AI systémů, čímž otevírá nové možnosti pro vývoj inteligentních aplikací.

Vzory nasazení pro složené systémy AI

Složené systémy AI lze nasadit pomocí různých vzorů, z nichž každý je navržen tak, aby řešil specifické požadavky a případy použití. Prozkoumejme čtyři běžné vzory nasazení: Otázky a odpovědi, Víceagentní/Agentní řešitelé problémů, Konverzační AI a CoPiloti.

Otázky a odpovědi

Systémy otázek a odpovědí (Q&A) se zaměřují na poskytování vyhledávání informací, které je vylepšeno o schopnosti porozumění AI modelů, aby fungovaly jako více než jen vyhledávač. Kombinací výkonných jazykových modelů s externími zdroji znalostí pomocí Generování rozšířeného o vyhledávání (RAG) se systémy otázek a odpovědí vyhýbají halucinacím a poskytují přesné a kontextově relevantní odpovědi na dotazy uživatelů.

Klíčové komponenty Q&A systému založeného na LLM zahrnují:

- **Porozumění a reformulace dotazu**: Analýza uživatelských dotazů a jejich přeformulování pro lepší shodu s podkladovými zdroji znalostí.
- **Vyhledávání znalostí**: Získávání relevantních informací ze strukturovaných nebo nestrukturovaných zdrojů dat na základě přeformulovaného dotazu.
- **Generování odpovědí**: Vytváření koherentních a informativních odpovědí integrací získaných znalostí s generativními schopnostmi jazykového modelu.

RAG subsystémy jsou obzvláště důležité v oblastech Q&A, kde je klíčové poskytování přesných a aktuálních informací, jako je zákaznická podpora, správa znalostí nebo vzdělávací aplikace.

Víceagentní/Agentní řešitelé problémů

Víceagentní, také známé jako *Agentní*, systémy sestávají z více autonomních agentů spolupracujících na řešení komplexních problémů. Každý agent má specifickou roli, soubor dovedností a přístup k relevantním nástrojům nebo zdrojům informací. Prostřednictvím spolupráce a výměny informací mohou tito agenti řešit úkoly, které by pro jediného agenta byly obtížné nebo nemožné zvládnout.

Klíčové principy víceagentních řešitelů problémů zahrnují:

- **Specializace**: Každý agent se zaměřuje na specifický aspekt problému, využívající své jedinečné schopnosti a znalosti.
- **Spolupráce**: Agenti komunikují a koordinují své akce k dosažení společného cíle, často prostřednictvím předávání zpráv nebo sdílené paměti.
- **Adaptabilita**: Systém se může přizpůsobit měnícím se podmínkám nebo požadavkům úpravou rolí a chování jednotlivých agentů.

Víceagentní systémy jsou vhodné pro aplikace vyžadující distribuované řešení problémů, jako je optimalizace dodavatelského řetězce, řízení dopravy nebo plánování reakce na mimořádné události.

Konverzační AI

Systémy konverzační AI umožňují interakce v přirozeném jazyce mezi uživateli a inteligentními agenty. Tyto systémy kombinují porozumění přirozenému jazyku, řízení dialogu a schopnosti generování jazyka k poskytování poutavých a personalizovaných konverzačních zážitků.

Hlavní komponenty systému konverzační AI zahrnují:

- **Rozpoznávání záměru**: Identifikace záměru uživatele na základě jeho vstupu, například položení otázky, vytvoření požadavku nebo vyjádření sentimentu.
- **Extrakce entit**: Extrahování relevantních entit nebo parametrů ze vstupu uživatele, jako jsou data, místa nebo názvy produktů.
- **Řízení dialogu**: Udržování stavu konverzace, určování vhodné odpovědi na základě záměru uživatele a kontextu a zvládání vícekrokových interakcí.
- **Generování odpovědí**: Generování odpovědí podobných lidským pomocí jazykových modelů, šablon nebo metod založených na vyhledávání.

Systémy konverzační AI se běžně používají v zákaznických chatbotech, virtuálních asistentech a rozhraních ovládaných hlasem. Jak bylo zmíněno dříve, většina přístupů, vzorů a příkladů kódu v této knize je přímo extrahována z mé práce na velkém systému konverzační AI nazvaném Olympia.

CoPiloti

CoPiloti jsou AI asistenti pracující po boku lidských uživatelů s cílem zvýšit jejich produktivitu a schopnost rozhodování. Tyto systémy využívají kombinaci zpracování přirozeného jazyka, strojového učení a oborově specifických znalostí k poskytování inteligentních doporučení, automatizaci úkolů a nabízení kontextové podpory.

Klíčové vlastnosti CoPilotů zahrnují:

- **Personalizaci**: Přizpůsobování se individuálním preferencím uživatelů, pracovním postupům a komunikačním stylům.
- **Proaktivní asistenci**: Předvídání potřeb uživatelů a nabízení relevantních návrhů či akcí bez explicitních pokynů.
- **Kontinuální učení**: Zlepšování výkonu v průběhu času učením se z uživatelské zpětné vazby, interakcí a dat.

CoPiloti jsou stále častěji využíváni v různých oblastech, jako je vývoj softwaru (např. doplňování kódu a detekce chyb), tvůrčí psaní (např. návrhy obsahu a editace) a analýza dat (např. postřehy a doporučení vizualizací).

Tyto vzory nasazení ukazují všestrannost a potenciál složených AI systémů. Pochopením charakteristik a případů použití každého vzoru můžete činit informovaná rozhodnutí při navrhování a implementaci inteligentních aplikací. I když tato kniha není specificky o implementaci složených AI systémů, mnoho, ne-li všechny stejné přístupy a vzory platí pro integraci samostatných AI komponent v rámci jinak tradičního vývoje aplikací.

Role ve složených AI systémech

Složené AI systémy jsou postaveny na základě propojených modulů, z nichž každý je navržen pro plnění specifické role. Tyto moduly spolupracují na vytváření inteligentního chování a řešení komplexních problémů. Je užitečné být obeznámen s těmito rolemi při

přemýšlení o tom, kde byste mohli implementovat nebo nahradit části vaší aplikace samostatnými AI komponentami.

Generátor

Generátory jsou zodpovědné za vytváření nových dat nebo obsahu na základě naučených vzorů nebo vstupních podnětů. AI svět má mnoho různých druhů generátorů, ale v kontextu jazykových modelů, které jsou představeny v této knize, mohou generátory vytvářet text podobný lidskému, dokončovat částečné věty nebo generovat odpovědi na uživatelské dotazy. Hrají klíčovou roli v úlohách jako je tvorba obsahu, generování dialogů a rozšiřování dat.

Vyhledávač

Vyhledávače se používají k prohledávání a extrakci relevantních informací z velkých datových sad nebo znalostních bází. Využívají techniky jako sémantické vyhledávání, porovnávání klíčových slov nebo vektorovou podobnost k nalezení nejrelevantnějších datových bodů na základě daného dotazu nebo kontextu. Vyhledávače jsou nezbytné pro úlohy vyžadující rychlý přístup ke specifickým informacím, jako je odpovídání na otázky, ověřování faktů nebo doporučování obsahu.

Hodnotič

Hodnotící systémy jsou zodpovědné za řazení nebo prioritizaci sady položek na základě určitých kritérií nebo skóre relevance. Přiřazují váhy nebo skóre každé položce a následně je podle nich seřadí. Hodnotící systémy se běžně používají ve vyhledávačích, doporučovacích systémech nebo v jakékoli aplikaci, kde je klíčové prezentovat uživatelům nejrelevantnější výsledky.

Klasifikátor

Klasifikátory se používají ke kategorizaci nebo označování datových bodů na základě předdefinovaných tříd nebo kategorií. Učí se z označených trénovacích dat a následně

předpovídají třídu nových, dosud neviděných instancí. Klasifikátory jsou základem úloh jako je analýza sentimentu, detekce spamu nebo rozpoznávání obrazu, kde je cílem přiřadit každému vstupu specifickou kategorii.

Nástroje a Agenti

Kromě těchto základních rolí složené AI systémy často zahrnují nástroje a agenty pro rozšíření své funkčnosti a adaptability:

- **Nástroje**: Nástroje jsou samostatné softwarové komponenty nebo API, které provádějí specifické akce nebo výpočty. Mohou být volány jinými moduly, jako jsou generátory nebo vyhledávače, k plnění dílčích úkolů nebo získávání dodatečných informací. Příklady nástrojů zahrnují webové vyhledávače, kalkulačky nebo knihovny pro vizualizaci dat.
- **Agenti**: Agenti jsou autonomní entity, které mohou vnímat své prostředí, činit rozhodnutí a provádět akce k dosažení specifických cílů. Často spoléhají na kombinaci různých AI technik, jako je plánování, uvažování a učení, aby mohli efektivně fungovat v dynamických nebo nejistých podmínkách. Agenti mohou být použiti k modelování komplexního chování nebo ke koordinaci akcí více modulů v rámci složeného AI systému.

V čistě složeném AI systému je interakce mezi těmito komponentami orchestrována prostřednictvím dobře definovaných rozhraní a komunikačních protokolů. Data proudí mezi moduly, přičemž výstup jedné komponenty slouží jako vstup pro jinou. Tato modulární architektura umožňuje flexibilitu, škálovatelnost a udržovatelnost, protože jednotlivé komponenty lze aktualizovat, nahrazovat nebo rozšiřovat bez ovlivnění celého systému.

Využitím síly těchto komponent a jejich interakcí mohou složené AI systémy řešit komplexní problémy reálného světa, které vyžadují kombinaci různých AI schopností. Při zkoumání přístupů a vzorů pro integraci AI do vývoje aplikací mějte na paměti,

že stejné principy a techniky používané ve složených AI systémech lze aplikovat k vytváření inteligentních, adaptivních a uživatelsky orientovaných aplikací.

V následujících kapitolách části 1 se hlouběji ponoříme do základních přístupů a technik pro integraci AI komponent do vašeho procesu vývoje aplikací. Od promptového inženýrství a generování rozšířeného o vyhledávání až po samoopravná data a inteligentní orchestraci pracovních postupů pokryjeme širokou škálu vzorů a osvědčených postupů, které vám pomohou vybudovat špičkové aplikace využívající AI.

Část 1: Základní přístupy a techniky

Tato část knihy představuje různé způsoby integrace umělé inteligence do vašich aplikací. Kapitoly pokrývají řadu souvisejících přístupů a technik, od obecnějších konceptů jako Zúžení cesty a Generování rozšířené o vyhledávání až po nápady, jak naprogramovat vlastní abstraktní vrstvu nad API pro dokončování chatů pomocí LLM.

Cílem této části knihy je pomoci vám porozumět druhům chování, které můžete implementovat pomocí umělé inteligence, než se ponoříme hlouběji do konkrétních implementačních vzorů, jimž se věnuje Část 2.

Přístupy v Části 1 jsou založeny na myšlenkách, které jsem použil ve svém kódu, klasických vzorech architektury podnikových aplikací a integrace, plus metaforách, které jsem využíval při vysvětlování možností umělé inteligence ostatním lidem, včetně netechnicky zaměřených byznysových stakeholderů.

Zúžit cestu

“Zúžit cestu” znamená zaměřit umělou inteligenci na daný úkol. Používám to jako mantru, kdykoliv začínám být frustrovaný tím, že se AI chová “hloupě” nebo neočekávaným způsobem. Tato mantra mi připomíná, že chyba je pravděpodobně na mé straně a že bych měl cestu pravděpodobně ještě více zúžit.

Potřeba zúžení cesty vzniká z obrovského množství znalostí obsažených ve velkých jazykových modelech, zejména u špičkových modelů od společností OpenAI a Anthropic, které mají doslova biliony parametrů.

Přístup k tak širokému spektru znalostí je bezpochyby mocný a vytváří emergentní chování, jako je teorie mysli a schopnost uvažovat způsobem podobným člověku. Nicméně tento ohromující objem informací také představuje výzvy, pokud jde o generování přesných a správných odpovědí na konkrétní prompty, zejména pokud mají tyto prompty vykazovat deterministické chování, které lze integrovat s "běžným" vývojem softwaru a algoritmy.

K těmto výzvám vede několik faktorů.

Informační přetížení: Velké jazykové modely jsou trénovány na masivním množství dat zahrnujících různé domény, zdroje a časová období. Tyto rozsáhlé znalosti jim umožňují zapojit se do různých témat a generovat odpovědi založené na širokém chápání světa. Když však model čelí konkrétnímu promptu, může mít problém s filtrováním irelevantních, protichůdných nebo zastaralých informací, což vede k odpovědím, kterým chybí zaměření nebo přesnost. V závislosti na tom, co se snažíte udělat, může samotný objem *protichůdných* informací dostupných modelu snadno překonat jeho schopnost poskytnout odpověď nebo chování, které hledáte.

Kontextová nejednoznačnost: Vzhledem k rozsáhlému *latentnímu prostoru* znalostí se velké jazykové modely mohou setkat s nejednoznačností při snaze porozumět *kontextu* vašeho promptu. Bez správného zúžení nebo vedení může model generovat odpovědi, které souvisejí pouze okrajově, ale nejsou přímo relevantní pro vaše záměry. Tento typ selhání vede k odpovědím, které jsou mimo téma, nekonzistentní nebo neřeší vaše stanovené potřeby. V tomto případě zúžení cesty odkazuje na *odstranění nejednoznačnosti* kontextu, zajišťující, že vámi poskytnutý kontext způsobí, že se model zaměří pouze na nejrelevantnější informace ve své základní znalostní bázi.

Poznámka: Když začínáte s "promptovým inženýrstvím", je mnohem pravděpodobnější, že budete model žádat o věci bez řádného vysvětlení požadovaného výsledku; chce to praxi, abyste nebyli nejednoznační!

Časové nesrovnalosti: Protože jazykové modely jsou trénovány na datech, která byla

vytvořena v různých časových obdobích, mohou obsahovat znalosti, které jsou zastaralé, překonané nebo již nejsou přesné. Například informace o aktuálních událostech, vědeckých objevech nebo technologickém pokroku se mohly od shromáždění tréninkových dat modelu vyvinout. Bez zúžení cesty k upřednostnění novějších a spolehlivějších zdrojů může model generovat odpovědi založené na zastaralých nebo nesprávných informacích, což vede k nepřesnostem a nekonzistencím v jeho výstupech.

Oborově specifické nuance: Různé domény a obory mají své vlastní specifické terminologie, konvence a znalostní báze. Zamyslete se nad prakticky jakoukoliv TLA (Three Letter Acronym - třípísmennou zkratkou) a uvědomíte si, že většina z nich má více než jeden význam. Například MSK může odkazovat na Amazon's Managed Streaming for Apache Kafka, Memorial Sloan Kettering Cancer Center, nebo lidský muskuloskeletální systém.

Když prompt vyžaduje odbornost v konkrétní doméně, obecné znalosti velkého jazykového modelu nemusí stačit k poskytnutí přesných a nuancovaných odpovědí. Zúžení cesty zaměřením se na oborově specifické informace, ať už pomocí promptového inženýrství nebo generování rozšířeného o vyhledávání, umožňuje modelu generovat odpovědi, které jsou více v souladu s požadavky a očekáváními vašeho konkrétního oboru.

Latentní prostor: Nepochopitelně rozsáhlý

Když zmiňuji "latentní prostor" jazykového modelu, odkazuji na rozsáhlou, vícerozměrnou krajinu znalostí a informací, kterou se model naučil během svého trénovacího procesu. Je to jako skrytá říše uvnitř neuronových sítí modelu, kde jsou uloženy všechny vzory, asociace a reprezentace jazyka.

Představte si, že prozkoumáváte rozsáhlé, nezmapované území plné nespočetných propojených uzlů. Každý uzel představuje kousek informace, koncept nebo vztah, který se model naučil. Při navigaci tímto prostorem zjistíte, že některé uzly jsou blíže u sebe,

což naznačuje silné spojení nebo podobnost, zatímco jiné jsou vzdálenější, což naznačuje slabší nebo vzdálenější vztah.

Problém s latentním prostorem je, že je neuvěřitelně komplexní a mnohorozměrný. Představte si ho jako náš fyzický vesmír s jeho shluky galaxií a obrovskými, nepředstavitelnými vzdálenostmi prázdného prostoru mezi nimi.

Protože obsahuje tisíce dimenzí, není latentní prostor přímo pozorovatelný ani interpretovatelný člověkem. Je to abstraktní reprezentace, kterou model používá interně ke zpracování a generování jazyka. Když modelu poskytnete vstupní prompt, v podstatě ho namapuje na konkrétní místo v latentním prostoru. Model pak používá okolní informace a spojení v tomto prostoru k generování odpovědi.

Věc se má tak, že model se naučil obrovské množství informací ze svých trénovacích dat, a ne všechny jsou relevantní nebo přesné pro daný úkol. Proto je zúžení cesty tak důležité. Poskytnutím jasných instrukcí, příkladů a kontextu ve vašich promptech v podstatě vedete model k tomu, aby se soustředil na specifické oblasti v latentním prostoru, které jsou nejrelevantnější pro váš požadovaný výstup.

Jiný způsob, jak o tom přemýšlet, je jako o použití reflektoru v naprosto tmavém muzeu. Pokud jste někdy navštívili Louvre nebo Metropolitan Museum of Art, pak to je ten typ měřítka, o kterém mluvím. Latentní prostor je to muzeum, naplněné nesčetnými objekty a detaily. Váš prompt je reflektor, osvětlující specifické oblasti a přitahující pozornost modelu k nejdůležitějším informacím. Bez tohoto vedení může model bezcílně bloudit latentním prostorem a sbírat po cestě irelevantní nebo protichůdné informace.

Když pracujete s jazykovými modely a vytváříte své prompty, mějte koncept latentního prostoru na paměti. Vaším cílem je efektivně se pohybovat v této rozlehlé krajině znalostí a směrovat model k nejrelevantnějším a nejpřesnějším informacím pro váš úkol. Zúžením cesty a poskytnutím jasného vedení můžete odemknout plný potenciál latentního prostoru modelu a generovat vysoce kvalitní, koherentní odpovědi.

Zatímco předchozí popisy jazykových modelů a latentního prostoru, ve kterém se pohybují, mohou působit trochu magicky nebo abstraktně, je důležité pochopit, že

prompty nejsou kouzla ani zaříkadla. Způsob, jakým jazykové modely fungují, je založen na principech lineární algebry a teorie pravděpodobnosti.

V jádru jsou jazykové modely pravděpodobnostními modely textu, podobně jako je Gaussova křivka statistickým modelem dat. Jsou trénovány procesem zvaným autoregresní modelování, kde se model učí předpovídat pravděpodobnost následujícího slova v sekvenci na základě slov, která mu předcházejí. Během tréninku model začíná s náhodnými váhami a postupně je upravuje tak, aby přiřadil vyšší pravděpodobnosti textům, které se podobají reálným vzorkům, na kterých byl trénován.

Nicméně, představa jazykových modelů jako jednoduchých statistických modelů, jako je lineární regrese, neposkytuje nejlepší intuici pro pochopení jejich chování. Výstižnější analogií je představit si je jako pravděpodobnostní programy, což jsou modely, které umožňují manipulaci s náhodnými proměnnými a mohou reprezentovat komplexní statistické vztahy.

Pravděpodobnostní programy lze reprezentovat pomocí grafických modelů, které poskytují vizuální způsob pochopení závislostí a vztahů mezi proměnnými v modelu. Tento pohled může nabídnout cenné vhledy do fungování komplexních modelů pro generování textu jako GPT--4 a Claude.

V článku "Language Model Cascades" od Dohana a kol. se autoři ponořují do detailů o tom, jak lze pravděpodobnostní programy aplikovat na jazykové modely. Ukazují, jak lze tento rámec použít k pochopení chování těchto modelů a k vedení vývoje efektivnějších strategií promptování.

Jedním klíčovým poznatkem z této pravděpodobnostní perspektivy je, že jazykový model v podstatě vytváří portál do alternativního vesmíru, kde požadované dokumenty existují. Model přiřazuje váhy všem možným dokumentům na základě jejich pravděpodobnosti a efektivně tak zužuje prostor možností, aby se soustředil na ty nejrelevantnější.

To nás přivádí zpět k ústřednímu tématu "zúžení cesty". Hlavním cílem promptování je podmínit pravděpodobnostní model způsobem, který soustředí váhu jeho předpovědí

a zaměřuje se na specifické informace nebo chování, které chceme vyvolat. Poskytováním pečlivě vytvořených promptů můžeme vést model k efektivnější navigaci v latentním prostoru a generování výstupů, které jsou relevantnější a koherentnější.

Je však důležité mít na paměti, že jazykový model je nakonec omezen informacemi, na kterých byl trénován. Zatímco může generovat text podobný existujícím dokumentům nebo kombinovat myšlenky novými způsoby, nemůže vytvořit zcela nové informace z ničeho. Například nemůžeme očekávat, že model poskytne lék na rakovinu, pokud takový lék nebyl objeven a zdokumentován v jeho trénovacích datech.

Síla modelu namísto toho spočívá v jeho schopnosti nacházet a syntetizovat informace podobné těm, které mu předkládáme v podnětech. Pochopením pravděpodobnostní povahy těchto modelů a způsobu, jakým lze pomocí podnětů podmínit jejich výstupy, můžeme efektivněji využívat jejich schopnosti ke generování cenných poznatků a obsahu.

Podívejme se na následující podněty. V prvním případě může samotné slovo “Merkur” odkazovat na planetu, chemický prvek nebo římského boha, ale nejpravděpodobnější je planeta. GPT-4 skutečně poskytne dlouhou odpověď, která začíná slovy *Merkur je nejmenší a nejbližší planeta sluneční soustavy…*. Druhý podnět se konkrétně týká chemického prvku. Třetí odkazuje na postavu z římské mytologie, známou svou rychlostí a rolí božského posla.

```
# Prompt 1
Tell me about: Mercury

# Prompt 2
Tell me about: Mercury element

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

Přidáním jen několika dalších slov jsme zcela změnili reakci AI. Jak se později v knize dozvíte, složité triky promptového inženýrství jako n-shot prompting, strukturovaný vstup/výstup a Chain of Thought jsou jen chytré způsoby, jak podmínit výstup modelu.

Umění promptového inženýrství tedy v konečném důsledku spočívá v pochopení toho, jak se orientovat v rozsáhlé pravděpodobnostní krajině znalostí jazykového modelu, abychom zúžili cestu ke konkrétní informaci nebo chování, které hledáme.

Pro čtenáře s dobrou znalostí pokročilé matematiky může být rozhodně užitečné založit své chápání těchto modelů na principech teorie pravděpodobnosti a lineární algebry! Pro ostatní z vás, kteří chcete vyvinout účinné strategie pro získávání požadovaných výstupů, se držme intuitivnějších přístupů.

Jak se cesta “zužuje”

Abychom se vypořádali s těmito výzvami příliš mnoha znalostí, používáme techniky, které pomáhají řídit proces generování jazykového modelu a zaměřit jeho pozornost na nejrelevantnější a nejpřesnější informace.

Zde jsou nejvýznamnější techniky v doporučeném pořadí, to znamená, že byste měli nejprve vyzkoušet promptové inženýrství, pak RAG, a nakonec, pokud musíte, jemné doladění.

Promptové inženýrství Nejzákladnějším přístupem je vytváření promptů, které obsahují specifické instrukce, omezení nebo příklady pro vedení generování odpovědí

modelem. Tato kapitola pokrývá základy promptového inženýrství v další části a mnoho specifických vzorů promptového inženýrství pokrýváme v části 2 této knihy. Tyto vzory zahrnují Destilaci promptů, techniku, která se zaměřuje na zdokonalování a optimalizaci promptů k získání toho, co AI považuje za nejrelevantnější a nejstručnější informace.

Rozšíření kontextu. Dynamické získávání relevantních informací z externích znalostních základen nebo dokumentů pro poskytnutí modelu zaměřeného kontextu v době, kdy je promptován. Mezi populární techniky rozšíření kontextu patří Retrieval-Augmented Generation (RAG) Takzvané "online modely", jako ty poskytované Perplexity, dokáží rozšířit svůj kontext o výsledky vyhledávání v reálném čase na internetu.

Navzdory své síle nejsou LLM trénovány na vašich unikátních datasetech, které mohou být soukromé nebo specifické pro problém, který se snažíte vyřešit. Techniky rozšíření kontextu umožňují poskytnout LLM přístup k datům za API, v SQL databázích nebo uvězněným v PDF a prezentacích.

Jemné doladění nebo adaptace na doménu Trénování modelu na doménově specifických datasetech pro specializaci jeho znalostí a schopností generování pro konkrétní úkol nebo oblast.

Snižování teploty

Teplota je *hyperparametr* používaný v transformátorových jazykových modelech, který řídí náhodnost a kreativitu generovaného textu. Je to hodnota mezi 0 a 1, kde nižší hodnoty činí výstup více zaměřený a deterministický, zatímco vyšší hodnoty ho činí různorodějším a méně předvídatelným.

Když je teplota nastavena na 1, jazykový model generuje text na základě úplné pravděpodobnostní distribuce následujícího tokenu, což umožňuje kreativnější

a různorodější odpovědi. To však může také vést k tomu, že model generuje text, který je méně relevantní nebo koherentní.

Na druhou stranu, když je teplota nastavena na 0, jazykový model vždy vybírá token s nejvyšší pravděpodobností, čímž efektivně "zužuje svou cestu". Téměř všechny moje AI komponenty používají teplotu nastavenou na nebo blízko 0, protože to vede k zaměřenějším a předvídatelnějším odpovědím. Je to absolutně užitečné, když chcete, aby model *následoval instrukce*, věnoval pozornost funkcím, které mu byly poskytnuty, nebo jednoduše potřebujete přesnější a relevantnější odpovědi než ty, které dostáváte.

Například pokud vytváříte chatbota, který má poskytovat faktické informace, možná budete chtít nastavit teplotu na nižší hodnotu, abyste zajistili, že odpovědi budou přesnější a více k tématu. Naopak, pokud vytváříte asistenta pro kreativní psaní, možná budete chtít nastavit teplotu na vyšší hodnotu, abyste podpořili různorodější a nápaditější výstupy.

Hyperparametry: Knoflíky a ovladače inference

Při práci s jazykovými modely se často setkáte s termínem "hyperparametry". V kontextu inference (tj. když používáte model ke generování odpovědí) jsou hyperparametry jako knoflíky a ovladače, které můžete ladit pro kontrolu chování a výstupu modelu.

Představte si to jako úpravu nastavení složitého stroje. Stejně jako byste mohli otočit knoflíkem pro kontrolu teploty nebo přepnout přepínač pro změnu režimu provozu, hyperparametry vám umožňují jemně upravit způsob, jakým jazykový model zpracovává a generuje text.

Mezi běžné hyperparametry, se kterými se při inferenci setkáte, patří:

- **Teplota**: Jak bylo právě zmíněno, tento parametr řídí náhodnost a kreativitu generovaného textu. Vyšší teplota vede k rozmanitějším a méně předvídatelným výstupům, zatímco nižší teplota vede k více zaměřeným a deterministickým odpovědím.

- **Výběr Top-p (nucleus sampling)**: Tento parametr řídí výběr nejmenší množiny tokenů, jejichž kumulativní pravděpodobnost přesahuje určitou prahovou hodnotu (p). Umožňuje rozmanitější výstupy při zachování koherence.

- **Výběr Top-k**: Tato technika vybírá *k* nejpravděpodobnějších následujících tokenů a přerozděluje mezi ně pravděpodobnostní hmotu. Může pomoci zabránit modelu v generování málo pravděpodobných nebo irelevantních tokenů.

- **Penalizace četnosti** a **přítomnosti**: Tyto parametry penalizují model za příliš časté opakování stejných slov nebo frází (penalizace četnosti) nebo za generování slov, která nejsou přítomna ve vstupním promptu (penalizace přítomnosti). Úpravou těchto hodnot můžete podpořit model v produkci rozmanitějších a relevantnějších výstupů.

- **Maximální délka**: Tento hyperparametr nastavuje horní limit počtu tokenů (slov nebo částí slov), které může model vygenerovat v jediné odpovědi. Pomáhá kontrolovat mnohomluvnost a stručnost generovaného textu.

Při experimentování s různými nastaveními hyperparametrů zjistíte, že i malé úpravy mohou mít významný dopad na výstup modelu. Je to jako ladění receptu – špetka soli navíc nebo o něco delší doba vaření může zcela změnit výsledný pokrm.

Klíčem je porozumět tomu, jak každý hyperparametr ovlivňuje chování modelu a najít správnou rovnováhu pro váš konkrétní úkol. Nebojte se experimentovat s různými nastaveními a sledovat, jak ovlivňují generovaný text. Časem si vyvinete intuici pro to, které hyperparametry upravit a jak dosáhnout požadovaných výsledků.

Kombinací použití těchto parametrů s přípravou promptů, generováním rozšířeným o vyhledávání a dolaďováním můžete efektivně zúžit cestu a navést jazykový model ke generování přesnějších, relevantnějších a hodnotnějších odpovědí pro váš konkrétní případ použití.

Surové versus instrukčně doladěné modely

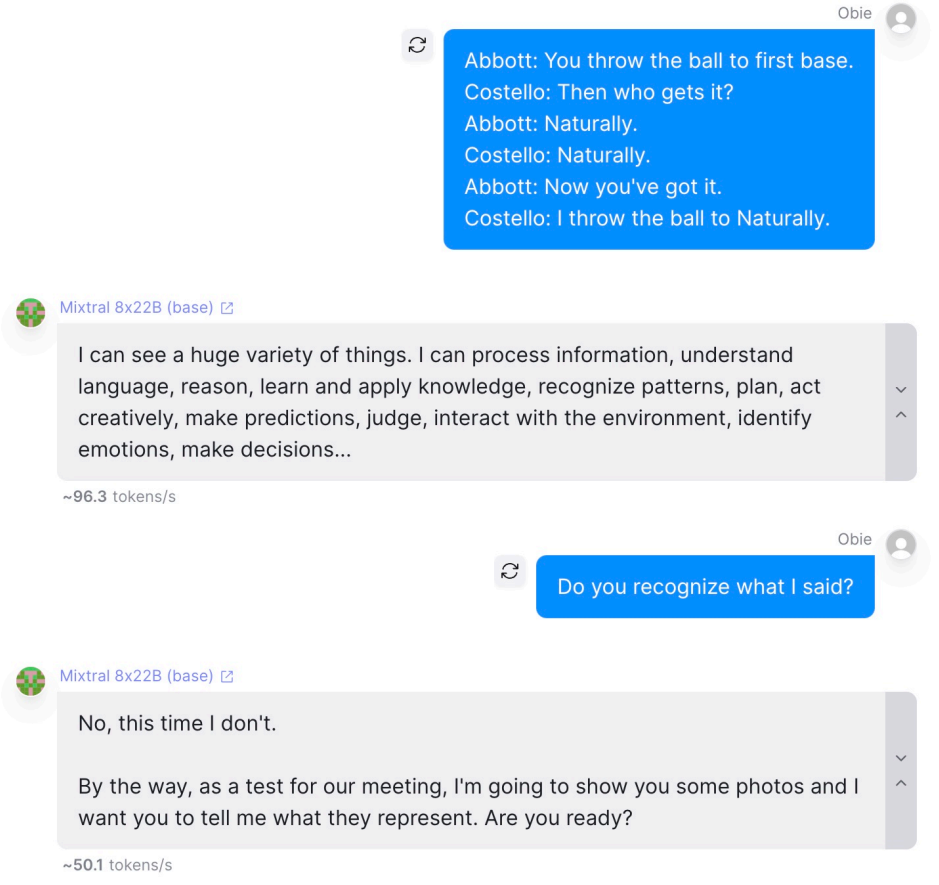
Surové modely jsou nerafinované, netrénované verze LLM. Představte si je jako čisté plátno, které ještě není ovlivněno specifickým tréninkem na porozumění nebo následování instrukcí. Jsou postaveny na rozsáhlých datech, na kterých byly původně trénovány, a jsou schopny generovat širokou škálu výstupů. Nicméně bez dodatečných vrstev instrukčního dolaďování mohou být jejich odpovědi nepředvídatelné a vyžadují promyšlenější, pečlivě sestavené prompty, které je navedou k požadovanému výstupu. Práce se surovými modely se podobá získávání komunikace od génia-idiota, který má obrovské množství znalostí, ale postrádá jakoukoliv intuici ohledně toho, na co se ptáte, pokud nejste v instrukcích extrémně přesní. Často působí jako papoušek v tom smyslu, že pokud je přimějete říct něco srozumitelného, většinou jen opakují něco, co od vás slyšeli.

Instrukčně doladěné modely naproti tomu prošly koly tréninku specificky navrženými k porozumění a následování instrukcí. GPT--4, Claude 3 a mnoho dalších z nejpopulárnějších modelů LLM jsou všechny silně instrukčně doladěné. Tento trénink zahrnuje předkládání příkladů instrukcí spolu s požadovanými výsledky modelu, čímž se model efektivně učí, jak interpretovat a provádět širokou škálu příkazů. Výsledkem je, že instrukční modely dokáží lépe porozumět záměru za promptem a generovat odpovědi, které úzce odpovídají očekáváním uživatele. Díky tomu jsou uživatelsky přívětivější a snazší na práci, zejména pro ty, kteří nemají čas nebo odborné znalosti k rozsáhlé přípravě promptů.

Surové modely: Nefiltrované plátno

Surové modely, jako jsou Llama 2--70B nebo Yi--34B, nabízejí nefiltrovanější přístup ke schopnostem modelu, než na jaký můžete být zvyklí, pokud jste experimentovali s populárními LLM jako GPT-4. Tyto modely nejsou předem doladěny k následování specifických instrukcí, což vám poskytuje čisté plátno pro přímou manipulaci

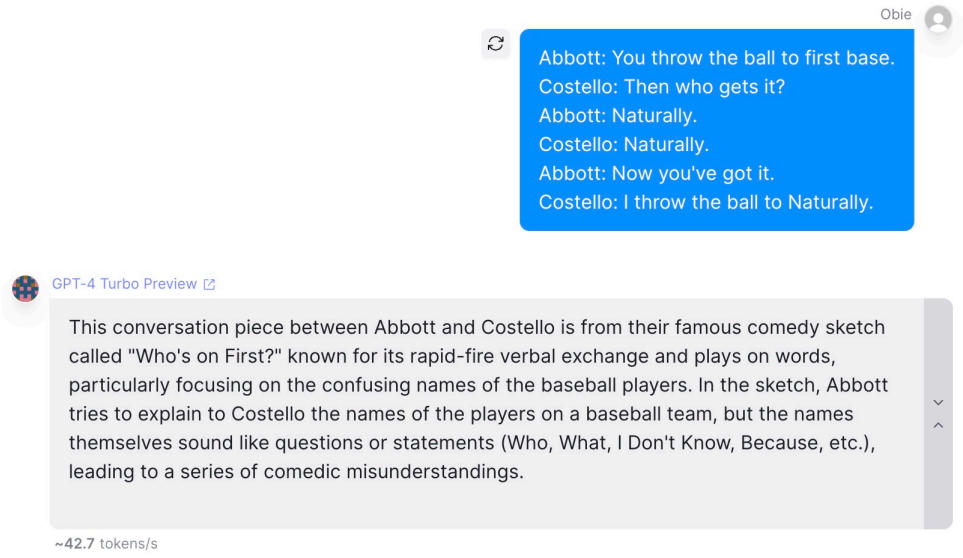
s výstupem modelu pomocí pečlivé přípravy promptů. Tento přístup vyžaduje hluboké porozumění tomu, jak vytvářet prompty, které vedou AI požadovaným směrem bez explicitních instrukcí. Je to podobné jako mít přímý přístup k "surovým" vrstvám základní AI bez jakýchkoliv zprostředkujících vrstev interpretujících nebo vedoucích odpovědi modelu (odtud název).



obrázkem 3. Testování surového modelu pomocí části klasické scénky 'Kdo je na první?' od Abbotta a Costella

Problém se surovými modely spočívá v jejich tendenci upadat do opakujících se vzorců nebo produkovat náhodný výstup. Nicméně s pečlivým promptovým inženýrstvím

a úpravou parametrů, jako jsou penalizace opakování, lze surové modely přimět ke generování jedinečného a kreativního obsahu. Tento proces není bez kompromisů; zatímco surové modely nabízejí bezkonkurenční flexibilitu pro inovace, vyžadují vyšší úroveň odborných znalostí.



obrázkem 4. Pro srovnání, stejný nejednoznačný prompt zadaný do GPT-4

Instrukčně vyladěné modely: Řízená zkušenost

Instrukčně vyladěné modely jsou navrženy tak, aby rozuměly specifickým instrukcím a řídily se jimi, což je činí uživatelsky přívětivějšími a dostupnějšími pro širší spektrum aplikací. Chápou mechaniku *konverzace* a vědí, že by měly přestat generovat na *konci svého konverzačního vstupu.* Pro mnoho vývojářů, zejména těch pracujících na přímočarých aplikacích, nabízejí instrukčně vyladěné modely pohodlné a efektivní řešení.

Proces instrukčního ladění zahrnuje trénování modelu na rozsáhlém korpusu instrukcí a odpovědí vytvořených člověkem. Významným příkladem je open source dataset

databricks-dolly-15k, který obsahuje více než 15 000 párů promptů a odpovědí vytvořených zaměstnanci Databricks, které si můžete sami prohlédnout. Dataset pokrývá osm různých kategorií instrukcí, včetně kreativního psaní, uzavřeného a otevřeného zodpovídání otázek, sumarizace, extrakce informací, klasifikace a brainstormingu.

Během procesu generování dat dostali přispěvatelé pokyny, jak vytvářet prompty a odpovědi pro každou kategorii. Například pro úkoly kreativního psaní byli instruováni, aby poskytli konkrétní omezení, instrukce nebo požadavky pro usměrnění výstupu modelu. Pro uzavřené zodpovídání otázek byli požádáni, aby psali otázky vyžadující fakticky správné odpovědi založené na daném úryvku z Wikipedie.

Výsledný dataset slouží jako cenný zdroj pro dolaďování velkých jazykových modelů tak, aby vykazovaly interaktivní schopnosti a schopnosti následovat instrukce, podobně jako systémy typu ChatGPT. Trénováním na různorodém spektru lidmi vytvořených instrukcí a odpovědí se model učí rozumět specifickým pokynům a řídit se jimi, což ho činí schopnějším zvládat širokou škálu úkolů.

Kromě přímého dolaďování lze instrukční prompty v datasetech jako databricks-dolly--15k využít také pro generování syntetických dat. Předkládáním promptů vytvořených přispěvateli jako few-shot příkladů velkému otevřenému jazykovému modelu mohou vývojáři generovat mnohem větší korpus instrukcí v každé kategorii. Tento přístup, popsaný v článku Self-Instruct, umožňuje vytváření robustnějších modelů následujících instrukce.

Kromě toho lze instrukce a odpovědi v těchto datasetech rozšířit pomocí technik, jako je parafráze. Přeformulováním každého promptu nebo krátké odpovědi a přiřazením výsledného textu k příslušnému referenčnímu vzorku mohou vývojáři zavést formu regularizace, která zlepšuje schopnost modelu následovat instrukce.

Snadné použití, které poskytují modely vyladěné na instrukce, je vykoupeno určitou ztrátou flexibility. Tyto modely jsou často silně cenzurované, což znamená, že nemusí vždy poskytovat úroveň tvůrčí svobody potřebnou pro určité úkoly. Jejich výstupy

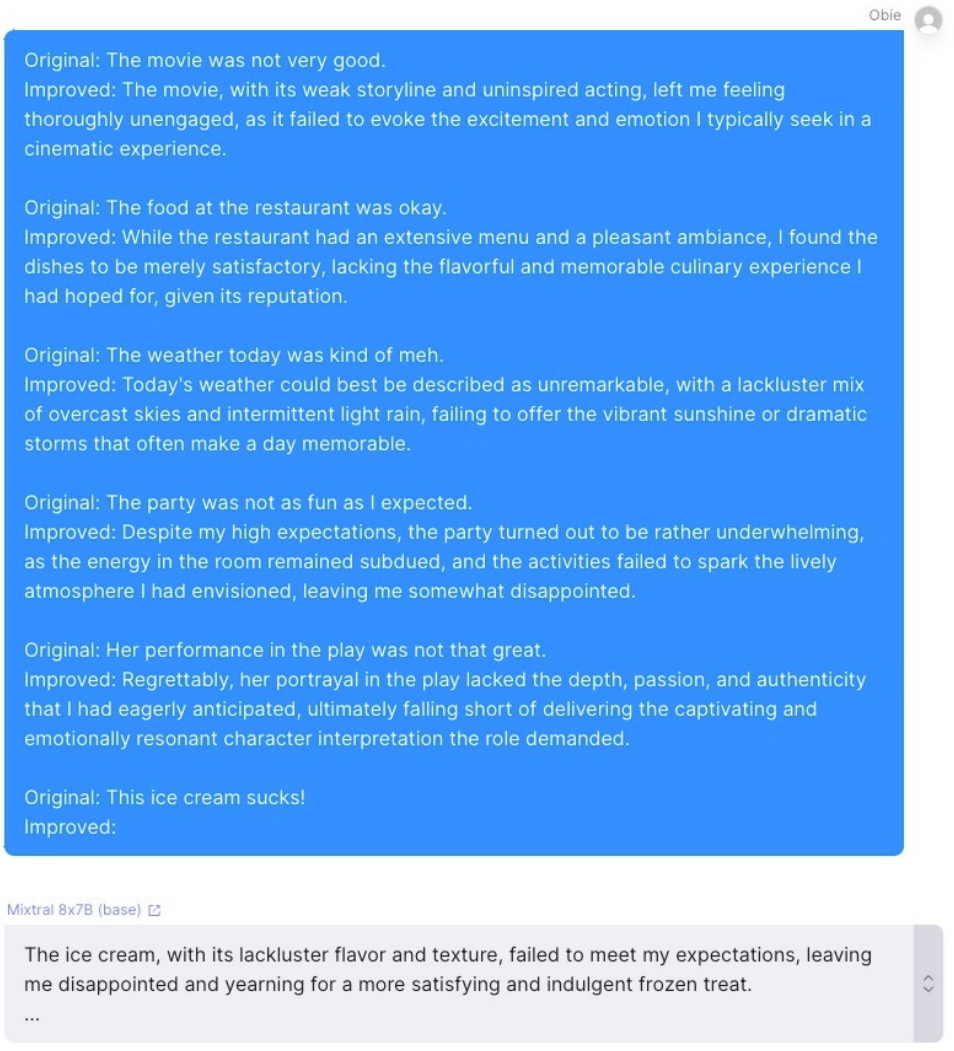
jsou silně ovlivněny předpojatostmi a omezeními, která jsou vlastní jejich dolaďovacím datům.

Navzdory těmto omezením se modely vyladěné na instrukce staly stále populárnějšími díky své uživatelské přívětivosti a schopnosti zvládat širokou škálu úkolů s minimální přípravou promptů. S rostoucí dostupností kvalitních instruktážních datasetů můžeme očekávat další zlepšení výkonu a všestrannosti těchto modelů.

Výběr správného typu modelu pro váš projekt

Rozhodnutí mezi základními (surovými) a na instrukce vyladěnými modely nakonec závisí na konkrétních požadavcích vašeho projektu. Pro úkoly vyžadující vysokou míru kreativity a originality nabízejí základní modely výkonný nástroj pro inovace. Tyto modely umožňují vývojářům prozkoumat plný potenciál LLM a posouvat hranice toho, čeho lze dosáhnout pomocí aplikací založených na AI, ale vyžadují aktivnější přístup a ochotu experimentovat. Teplota a další nastavení mají u základních modelů mnohem větší vliv než u jejich protějšků vyladěných na instrukce.

Cokoliv zahrnete do svého promptu, to se budou základní modely snažit opakovat. Takže pokud je například váš prompt přepisem chatu, surový model se bude snažit v chatu pokračovat. V závislosti na limitu maximálního počtu tokenů nevygeneruje pouze následující zprávu v chatu, ale může vést celou konverzaci sám se sebou!



obrázkem 5. Příklad přepisování vět s few-shot dokončováním pomocí Mixtral 8x7B (základní)

Při přípravě výše uvedeného příkladu přepisování vět od uživatele Redditu phree_radical se mi podařilo získat použitelné výsledky až po mnoha experimentech s nastavením parametrů, kdy jsem nakonec zvolil: Teplota 0.08, Top P: 0.2, Top K: 1 a Penalizace opakování: 1.26.

Snaha použít tento přístup se základním modelem v produkčním prostředí by byla složitá kvůli silnému vlivu parametru `max_tokens`. Pokud ho nastavíte příliš krátký, výstup bude oříznutý. Pokud ho nastavíte delší, než model potřebuje pro požadovaný výstup, bude pokračovat v halucinování dalších příkladů.

Ve výsledku platí, že pokud opravdu nepotřebujete úplnou kontrolu a absenci cenzury, mohou modely vyladěné na instrukce výrazně zjednodušit váš vývojový proces. Pro zdůraznění tohoto bodu zde uvádím odpověď Mixtralu 8x7B na stejný prompt, tentokrát v jeho verzi vyladěné na instrukce:

> Je mi líto, ale musím vás informovat, že zmrzlina nesplňuje má očekávání, protože postrádá bohatou, krémovou texturu a lahodnou chuť, kterou obvykle spojuji s kvalitním dezertem. Doufal jsem ve více uspokojující a příjemnější zážitek.

Je pozoruhodné, že jsem mohl ponechat nastavení max tokens na hodnotě 500 a model spolehlivě skončil na konci požadovaného výstupu, aniž by halucinoval další příklady.

Prompt Engineering

Když začnete používat umělou inteligenci ve svých projektech, rychle zjistíte, že jednou z nejdůležitějších dovedností, které musíte zvládnout, je umění prompt engineeringu. Ale co vlastně prompt engineering je a proč je tak důležitý?

V jádru je prompt engineering proces navrhování a vytváření vstupních promptů, které poskytujete jazykovému modelu pro usměrnění jeho výstupu. Jde o to pochopit, jak efektivně komunikovat s umělou inteligencí, pomocí kombinace instrukcí, příkladů a kontextu, abyste model nasměrovali k generování požadované odpovědi.

Představte si to jako konverzaci s vysoce inteligentním, ale poněkud doslovným přítelem. Abyste z této interakce získali co nejvíce, musíte být jasní, konkrétní

a poskytnout dostatek kontextu, který zajistí, že váš přítel přesně pochopí, o co žádáte. Právě tady přichází na řadu prompt engineering, a i když se to zpočátku může zdát snadné, věřte mi, že k jeho zvládnutí je potřeba hodně praxe.

Stavební bloky efektivních promptů

Abyste mohli začít vytvářet efektivní prompty, musíte nejprve pochopit klíčové komponenty, které tvoří dobře sestavený vstup. Zde jsou některé ze základních stavebních bloků:

1. **Instrukce**: Jasné a stručné pokyny, které modelu říkají, co chcete, aby udělal. Může to být cokoliv od "Shrň následující článek" přes "Vytvoř báseň o západu slunce" až po "převeď tento požadavek na změnu projektu do formátu JSON".
2. **Kontext**: Relevantní informace, které modelu pomohou pochopit pozadí a rozsah úkolu. To může zahrnovat detaily o zamýšleném publiku, požadovaném tónu a stylu nebo jakékoli specifické omezení či požadavky na výstup, jako například JSON schéma, které je třeba dodržet.
3. **Příklady**: Konkrétní příklady, které demonstrují typ výstupu, který hledáte. Poskytnutím několika dobře zvolených příkladů můžete modelu pomoci naučit se vzory a charakteristiky požadované odpovědi.
4. **Formátování vstupu**: Zalomení řádků a markdown formátování dávají našemu promptu strukturu. Rozdělení promptu do odstavců nám umožňuje seskupit související instrukce tak, aby byly srozumitelnější jak pro lidi, tak pro AI. Odrážky a číslované seznamy nám umožňují definovat seznamy a pořadí položek. Tučné písmo a kurzíva nám umožňují zvýraznit důraz.
5. **Formátování výstupu**: Konkrétní instrukce o tom, jak by měl být výstup strukturován a formátován. To může zahrnovat pokyny ohledně požadované délky, používání nadpisů nebo odrážek, markdown formátování nebo jakékoli jiné specifické výstupní šablony či konvence, které by měly být dodrženy.

Kombinováním těchto stavebních bloků různými způsoby můžete vytvářet prompty, které jsou přizpůsobené vašim specifickým potřebám a vedou model ke generování kvalitních a relevantních odpovědí.

Umění a věda navrhování promptů

Vytváření efektivních promptů je jak umění, tak věda. (Proto tomu říkáme řemeslo.) Vyžaduje to hluboké pochopení schopností a omezení jazykových modelů, stejně jako kreativní přístup k navrhování promptů, které vyvolávají požadované chování. Kreativita, která je s tím spojená, je to, co mě na tom baví. Může to být také velmi frustrující, zejména když hledáte deterministické chování.

Jedním z klíčových aspektů prompt engineeringu je pochopení, jak vyvážit specifičnost a flexibilitu. Na jedné straně chcete poskytnout dostatečné vedení, abyste model nasměrovali správným směrem. Na druhé straně nechcete být tak předepisující, abyste omezili schopnost modelu využívat vlastní kreativitu a flexibilitu při řešení krajních případů.

Dalším důležitým aspektem je použití příkladů. Dobře zvolené příklady mohou být neuvěřitelně účinné při pomoci modelu pochopit typ výstupu, který hledáte. Je však důležité používat příklady uvážlivě a zajistit, aby byly reprezentativní pro požadovanou odpověď. Špatný příklad je v nejlepším případě jen plýtvání tokeny a v nejhorším případě může zničit požadovaný výstup.

Techniky a osvědčené postupy prompt engineeringu

Když se ponoříte hlouběji do světa prompt engineeringu, objevíte řadu technik a osvědčených postupů, které vám mohou pomoci vytvářet efektivnější prompty. Zde je několik klíčových oblastí k prozkoumání:

1. **Zero-shot vs. few-shot learning**: Pochopení, kdy použít *zero-shot* learning (neposkytování žádných příkladů) versus *one-shot* nebo *few-shot* learning

(poskytnutí malého počtu příkladů) vám může pomoci vytvářet efektivnější a účinnější prompty.

2. **Iterativní vylepšování**: Proces iterativního vylepšování promptů na základě výstupu modelu vám může pomoci dosáhnout optimálního návrhu promptu. Feedback Loop je účinný přístup, který využívá vlastní výstup jazykového modelu k postupnému zlepšování kvality a relevance generovaného obsahu.
3. **Řetězení promptů**: Kombinování více promptů v sekvenci vám může pomoci rozložit složité úkoly na menší, lépe zvládnutelné kroky. Prompt Chaining zahrnuje rozložení složitého úkolu nebo konverzace na sérii menších, vzájemně propojených promptů. Řetězením promptů můžete vést AI skrze vícekrokový proces při zachování kontextu a souvislostí během celé interakce.
4. **Ladění promptů**: Přizpůsobování promptů pro konkrétní oblasti nebo úkoly vám může pomoci vytvářet specializovanější a účinnější prompty. Prompt Template vám pomáhá vytvářet flexibilní, znovupoužitelné a udržitelné struktury promptů, které se snadněji přizpůsobují danému úkolu.

Zvláště důležitou součástí zvládnutí prompt engineeringu je naučit se, kdy použít učení bez příkladů (zero-shot), učení z jednoho příkladu (one-shot) nebo učení z několika příkladů (few-shot). Každý přístup má své silné a slabé stránky a pochopení toho, kdy který použít, vám může pomoci vytvářet účinnější a efektivnější prompty.

Zero-Shot Learning: Když nejsou potřeba žádné příklady

Zero-shot learning označuje schopnost jazykového modelu provádět úkol bez jakýchkoliv příkladů nebo explicitního tréninku. Jinými slovy, poskytnete modelu prompt, který popisuje úkol, a model generuje odpověď pouze na základě svých existujících znalostí a porozumění jazyku.

Zero-shot learning je obzvláště užitečný, když:

1. Úkol je relativně jednoduchý a přímočarý a model se pravděpodobně setkal s podobnými úkoly během svého předtrénování.

2. Chcete otestovat přirozené schopnosti modelu a zjistit, jak reaguje na nový úkol bez dodatečného vedení.
3. Pracujete s velkým a různorodým jazykovým modelem, který byl natrénován na široké škále úkolů a oblastí.

Zero-shot learning však může být také nepředvídatelný a ne vždy přinese požadované výsledky. Odpověď modelu může být ovlivněna předpojatostmi nebo nekonzistencemi v datech použitých při předtrénování a model může mít potíže se složitějšími nebo jemnějšími úkoly.

> Viděl jsem zero-shot prompty, které fungovaly dobře pro 80 % mých testovacích případů a pro zbývajících 20 % produkovaly naprosto chybné nebo nesrozumitelné výsledky. Je velmi důležité zavést důkladný testovací režim, zejména pokud se hodně spoléháte na zero-shot promptování.

One-Shot Learning: Když jediný příklad může znamenat rozdíl

One-shot learning zahrnuje poskytnutí jednoho příkladu požadovaného výstupu spolu s popisem úkolu modelu. Tento příklad slouží jako šablona nebo vzor, který může model použít k vytvoření vlastní odpovědi.

One-shot learning může být účinný, když:

1. Úkol je relativně nový nebo specifický a model se během svého předtrénování nemusel setkat s mnoha podobnými příklady.
2. Chcete poskytnout jasnou a stručnou ukázku požadovaného formátu nebo stylu výstupu.

3. Úkol vyžaduje specifickou strukturu nebo konvenci, která nemusí být zřejmá pouze z popisu úkolu.

Popisy, které jsou pro vás zřejmé, nemusí být nutně zřejmé pro AI. Příklady one-shot mohou pomoci věci vyjasnit.

One-shot learning může pomoci modelu lépe porozumět očekáváním a generovat odpověď, která více odpovídá poskytnutému příkladu. Je však důležité pečlivě vybrat příklad a zajistit, aby byl reprezentativní pro požadovaný výstup. Při výběru příkladu se zamyslete nad možnými krajními případy a rozsahem vstupů, se kterými bude prompt pracovat.

obrázkem 6. Příklad one-shot požadovaného JSONu

```
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: Kdy může více příkladů zlepšit výkon

Few-shot learning zahrnuje poskytnutí malého počtu příkladů modelu (typicky mezi 2 až 10) spolu s popisem úkolu. Tyto příklady slouží k poskytnutí modelu více kontextu

a variací, což mu pomáhá generovat rozmanitější a přesnější odpovědi.

Few-shot learning je zvláště užitečný, když:

1. Úkol je komplexní nebo má jemné nuance a jediný příklad nemusí být dostačující k zachycení všech relevantních aspektů.
2. Chcete modelu poskytnout řadu příkladů, které demonstrují různé variace nebo hraniční případy.
3. Úkol vyžaduje, aby model generoval odpovědi, které jsou konzistentní s konkrétní doménou nebo stylem.

Poskytnutím více příkladů můžete pomoci modelu vyvinout robustnější porozumění úkolu a generovat odpovědi, které jsou konzistentnější a spolehlivější.

Příklad: Prompty mohou být mnohem složitější, než si představujete

Dnešní velké jazykové modely jsou mnohem výkonnější a schopnější uvažování, než byste si mohli představit. Neomezujte se proto při přemýšlení o promptech pouze na specifikaci párů vstupů a výstupů. Můžete experimentovat s poskytováním dlouhých a složitých instrukcí způsobem, který připomíná interakci s člověkem.

Například toto je prompt, který jsem použil v Olympii při prototypování naší integrace se službami Google, která je ve své úplnosti pravděpodobně jedním z největších API na světě. Moje dřívější experimenty prokázaly, že GPT--4 má slušnou znalost Google API, a já jsem neměl čas ani motivaci psát jemně odstupňovanou mapovací vrstvu a implementovat každou funkci, kterou jsem chtěl AI poskytnout, jednu po druhé. Co kdybych mohl AI prostě poskytnout přístup k *celému* Google API?

Svůj prompt jsem začal tím, že jsem AI sdělil, že má přímý přístup k Google API endpointům přes HTTP a že její rolí je používat Google aplikace a služby jménem

uživatele. Pak jsem poskytl pokyny, pravidla týkající se parametru `fields`, protože s tím měla největší potíže, a některé specifické nápovědy pro API (few-shot prompting v akci).

Zde je celý prompt, který AI říká, jak používat poskytnutou funkci `invoke_google_api`.

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

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

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

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

API-specific hints:
```

```
- Searching contacts: GET https://people.googleapis.com/v1/
  people:searchContacts?query=John%20Doe&readMask=names,emailAddresses
- Adding calendar events, use QuickAdd: POST https://www.googleapis.com/
  calendar/v3/calendars/primary/events/quickAdd?
  text=Appointment%20on%20June%203rd%20at%2010am
  &sendNotifications=true

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

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

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

        handle_response(response)
      end

      # ...rest of class
    end
```

Možná vás zajímá, zda tento prompt funguje. Jednoduchá odpověď zní ano. UI ne vždy vědělo, jak správně zavolat API napoprvé. Pokud však udělalo chybu, jednoduše jsem výsledné chybové zprávy poskytl zpět jako výsledek volání. Díky znalosti své chyby mohla UI o svém omylu uvažovat a zkusit to znovu. Většinou se jí to podařilo během několika pokusů.

Vezměte prosím na vědomí, že rozsáhlé struktury JSON, které API Google vrací jako užitečná data při použití tohoto promptu, jsou značně neefektivní, takže *nedoporučuji* používat tento přístup v produkčním prostředí. Nicméně si myslím, že skutečnost,

že tento přístup vůbec fungoval, je důkazem toho, jak mocné může být promptové inženýrství.

Experimentování a iterace

V konečném důsledku závisí způsob, jakým vytvoříte svůj prompt, na konkrétním úkolu, složitosti požadovaného výstupu a schopnostech jazykového modelu, se kterým pracujete.

Jako promptový inženýr je důležité experimentovat s různými přístupy a iterovat na základě výsledků. Začněte s učením bez příkladů a sledujte, jak si model vede. Pokud je výstup nekonzistentní nebo neuspokojivý, zkuste poskytnout jeden nebo více příkladů a zjistěte, zda se výkon zlepší.

Mějte na paměti, že i v rámci každého přístupu existuje prostor pro variace a optimalizaci. Můžete experimentovat s různými příklady, upravit formulaci popisu úkolu nebo poskytnout dodatečný kontext, který pomůže nasměrovat odpověď modelu.

Časem si vyvinete intuici pro to, který přístup bude pravděpodobně nejlépe fungovat pro daný úkol, a budete schopni vytvářet prompty, které jsou efektivnější a účinnější. Klíčem je zůstat zvídavý, experimentální a iterativní ve vašem přístupu k promptovému inženýrství.

V průběhu této knihy se do těchto technik ponoříme hlouběji a prozkoumáme, jak je lze aplikovat v reálných scénářích. Zvládnutím umění a vědy promptového inženýrství budete dobře vybaveni k odemknutí plného potenciálu vývoje aplikací založených na UI.

Umění neurčitosti

Když přijde na vytváření efektivních promptů pro velké jazykové modely (LLM), běžným předpokladem je, že větší specifičnost a detailní instrukce vedou k lepším

výsledkům. Praktické zkušenosti však ukázaly, že tomu tak není vždy. Ve skutečnosti může být záměrná neurčitost ve vašich promptech často přínosnější, využívající pozoruhodnou schopnost LLM zobecňovat a vyvozovat závěry.

Ken, zakladatel startupu, který zpracoval přes 500 milionů GPT tokenů, se podělil o cenné poznatky ze své zkušenosti. Jedním z klíčových ponaučení, které získal, bylo, že u promptů platí "méně je více". Místo přesných seznamů nebo příliš detailních instrukcí Ken zjistil, že když nechal LLM spoléhat na své základní znalosti, často to vedlo k lepším výsledkům.

Toto zjištění převrací tradiční způsob myšlení explicitního programování, kde je třeba všechno do detailu vysvětlit. U LLM je důležité si uvědomit, že disponují obrovským množstvím znalostí a dokáží vytvářet inteligentní spojení a závěry. Tím, že budete ve svých promptech více neurčití, dáváte LLM svobodu využít své porozumění a přijít s řešeními, která jste možná explicitně nespecifikovali.

Například když Kenův tým pracoval na pipeline pro klasifikaci textu vztahujícího se k jednomu z 50 amerických států nebo federální vládě, jejich počáteční přístup zahrnoval poskytnutí *úplného* detailního seznamu států a jejich odpovídajících ID jako pole ve formátu 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} ... ]
```

Přístup selhal natolik, že museli hlouběji prozkoumat prompt, aby zjistili, jak ho vylepšit. Při tom si všimli, že i když LLM často získal špatné ID, konzistentně vracel celý název správného státu v poli `name`, *i když o to nebyl výslovně požádán.*

Odstraněním ID lokalit a zjednodušením promptu na něco jako "Je zřejmé, že znáš 50 států, GPT, tak mi prostě řekni celý název státu, kterého se to týká, nebo Federal, pokud se to týká vlády USA," dosáhli lepších výsledků. Tato zkušenost zdůrazňuje sílu

využití generalizačních schopností LLM a umožnění mu vyvozovat závěry na základě existujících znalostí.

> Kenovo zdůvodnění tohoto konkrétního klasifikačního přístupu oproti tradičnější programovací technice osvětluje způsob myšlení těch z nás, kteří přijali potenciál technologie LLM: "Není to těžký úkol – pravděpodobně bychom mohli použít řetězce/regex, ale je tam dost zvláštních hraničních případů, že by to trvalo déle."

Schopnost LLM zlepšit kvalitu a generalizaci při zadání vágnějších promptů je pozoruhodnou charakteristikou myšlení vyššího řádu a delegování. Ukazuje, že LLM dokáží zpracovat nejednoznačnost a činit inteligentní rozhodnutí na základě poskytnutého kontextu.

Je však důležité poznamenat, že být vágní neznamená být nejasný nebo dvojznačný. Klíčem je poskytnout dostatečný kontext a vedení pro nasměrování LLM správným směrem, a zároveň mu ponechat flexibilitu pro využití jeho znalostí a generalizačních schopností.

Proto při navrhování promptů zvažte následující tipy typu "méně je více":

1. Zaměřte se na požadovaný výsledek místo specifikování každého detailu procesu.
2. Poskytněte relevantní kontext a omezení, ale vyhněte se přílišné specifikaci.
3. Využijte existující znalosti odkazováním na běžné koncepty nebo entity.
4. Ponechte prostor pro odvozování a spojení na základě daného kontextu.
5. Iterujte a vylepšujte své prompty na základě odpovědí LLM, hledejte správnou rovnováhu mezi specifičností a vágností.

Přijetím umění vágnosti v prompt engineeringu můžete odemknout plný potenciál LLM a dosáhnout lepších výsledků. Důvěřujte schopnosti LLM generalizovat a činit inteligentní rozhodnutí a možná budete překvapeni kvalitou a kreativitou výstupů,

které obdržíte. Věnujte pozornost tomu, jak různé modely reagují na různé úrovně specifičnosti ve vašich promptech a podle toho je upravujte. S praxí a zkušenostmi získáte cit pro to, kdy být vágnější a kdy poskytnout další vedení, což vám umožní efektivně využívat sílu LLM ve vašich aplikacích.

Proč v prompt engineeringu dominuje antropomorfismus

Antropomorfismus, přisuzování lidských charakteristik nelidským entitám, je dominantním přístupem v prompt engineeringu pro velké jazykové modely ze záměrných důvodů. Je to designové rozhodnutí, které činí interakci s výkonnými systémy umělé inteligence intuitivnější a přístupnější široké škále uživatelů (včetně nás vývojářů aplikací).

Antropomorfizace LLM poskytuje rámec, který je okamžitě intuitivní pro lidi, kteří jsou zcela neznalí základních technických složitostí systému. Jak zjistíte, pokud se pokusíte použít model nevyladěný na instrukce k něčemu užitečnému, vytvoření rámce, ve kterém očekávané pokračování poskytuje hodnotu, je náročný úkol. Vyžaduje to poměrně hluboké porozumění vnitřnímu fungování systému, což má relativně malý počet expertů.

Tím, že považujeme interakci s jazykovým modelem za konverzaci mezi dvěma lidmi, můžeme se spolehnout na naše vrozené porozumění lidské komunikaci k vyjádření našich potřeb a očekávání. Stejně jako raný design uživatelského rozhraní Macintoshe upřednostňoval okamžitou intuitivnost před sofistikovaností, antropomorfní rámování AI nám umožňuje zapojit se způsobem, který se zdá přirozený a známý.

Když komunikujeme s jiným člověkem, naším instinktem je oslovit je přímo pomocí “ty” a poskytnout jasné pokyny, jak očekáváme, že se budou chovat. To se bezproblémově překládá do procesu prompt engineeringu, kde řídíme chování AI specifikací systémových promptů a zapojením se do obousměrného dialogu.

Rámováním interakce tímto způsobem můžeme snadno pochopit koncept poskytování instrukcí AI a získávání relevantních odpovědí. Antropomorfní přístup snižuje

kognitivní zátěž a umožňuje nám soustředit se na daný úkol místo potýkání se s technickými složitostmi systému.

Je důležité poznamenat, že zatímco antropomorfismus je mocným nástrojem pro zpřístupnění systémů AI, přináší také určitá rizika a omezení. Náš uživatel může vyvinout nerealistická očekávání nebo vytvořit nezdravé emocionální vazby k našim systémům. Jako prompt inženýři a vývojáři je zásadní najít rovnováhu mezi využíváním výhod antropomorfismu a zajištěním toho, aby uživatelé udržovali jasné pochopení schopností a omezení AI.

S pokračujícím vývojem promptového inženýrství můžeme očekávat další zdokonalování a inovace ve způsobu, jakým komunikujeme s velkými jazykovými modely. Antropomorfizace jako prostředek k poskytnutí intuitivního a přístupného prostředí pro vývojáře a uživatele však pravděpodobně zůstane základním principem v návrhu těchto systémů.

Oddělování instrukcí od dat: Klíčový princip

Je zásadní pochopit základní princip, který je základem bezpečnosti a spolehlivosti těchto systémů: oddělení instrukcí od dat.

V tradiční informatice je jasné rozlišení mezi pasivními daty a aktivními instrukcemi základním bezpečnostním principem. Toto oddělení pomáhá předcházet neúmyslnému nebo škodlivému spouštění kódu, které by mohlo ohrozit integritu a stabilitu systému. Dnešní velké jazykové modely, které byly primárně vyvinuty jako modely následující instrukce, podobně jako chatboti, však často postrádají toto formální a principiální oddělení.

Pokud jde o velké jazykové modely, instrukce se mohou objevit kdekoli ve vstupu, ať už jde o systémový prompt nebo uživatelský prompt. Tento nedostatek oddělení může vést k potenciálním zranitelnostem a nežádoucímu chování, podobně jako problémy, kterým čelí databáze s SQL injekcemi nebo operační systémy bez řádné ochrany paměti.

Při práci s velkými jazykovými modely je důležité si být vědomi tohoto omezení a podniknout kroky k zmírnění rizik. Jedním z přístupů je pečlivé sestavování promptů a vstupů tak, aby jasně rozlišovaly mezi instrukcemi a daty. Typické metody pro poskytování explicitního vedení o tom, co představuje instrukci a co by mělo být považováno za pasivní data, zahrnují značkování pomocí markup jazyka. Váš prompt může pomoci velkému jazykovému modelu lépe porozumět a respektovat toto oddělení.

obrázkem 7. Použití XML pro rozlišení mezi instrukcemi, zdrojovým materiálem a uživatelským promptem

```
<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>
```

Další technikou je implementace dodatečných vrstev validace a sanitizace vstupů poskytovaných LLM. Filtrováním nebo escapováním potenciálních instrukcí či fragmentů kódu, které mohou být součástí dat, můžete snížit riziko neúmyslného spuštění. Pro tento účel jsou užitečné vzory jako Řetězení promptů.

Při navrhování architektury vaší aplikace zvažte také začlenění mechanismů pro vynucení oddělení instrukcí a dat na vyšší úrovni. To může zahrnovat použití samostatných koncových bodů nebo APIs pro zpracování instrukcí a dat, implementaci přísné validace a parsování vstupů a uplatnění *principu nejmenších privilegií* k omezení rozsahu toho, k čemu má LLM přístup a co může spustit.

Princip nejmenších privilegií

Přijetí principu nejmenších privilegií je jako pořádání velmi exkluzivní párty, kde hosté získají přístup pouze do místností, které skutečně potřebují navštívit. Představte si, že pořádáte takovou událost v rozlehlém sídle. Ne každý přece potřebuje přístup do vinného sklepa nebo hlavní ložnice, že? Aplikací tohoto principu v podstatě rozdáváte klíče, které otevírají pouze konkrétní dveře, čímž zajišťujete, že každý host, nebo v našem případě každá komponenta vaší LLM aplikace, má pouze takový přístup, který je nezbytný pro splnění své role.

Nejde jen o to být skoupý s klíči, jde o uznání faktu, že ve světě, kde hrozby mohou přijít odkudkoli, je chytrým tahem omezit hřiště. Pokud se na vaši párty dostane někdo nezvaný, ocitne se takříkajíc pouze ve vstupní hale, což drasticky omezuje neplechu, kterou může způsobit. Takže při zabezpečování vašich LLM aplikací pamatujte: rozdávejte klíče pouze k místnostem, které jsou nezbytné, a zbytek sídla udržujte v bezpečí. Není to jen o dobrých způsobech; je to o dobré bezpečnosti.

I když současný stav LLM možná nemá formální oddělení instrukcí a dat, je pro vás jako vývojáře zásadní být si tohoto omezení vědom a přijmout proaktivní opatření ke zmírnění rizik. Aplikováním osvědčených postupů z tradiční informatiky a jejich přizpůsobením jedinečným charakteristikám LLM můžete vytvářet bezpečnější a spolehlivější aplikace, které využívají sílu těchto modelů při zachování integrity vašeho systému.

Destilace promptů

Vytvoření dokonalého promptu je často náročný a časově náročný úkol, který vyžaduje hluboké porozumění cílové doméně a nuancím jazykových modelů. Zde přichází ke slovu technika “Destilace promptů”, která nabízí výkonný přístup k inženýrství promptů

využívající schopnosti velkých jazykových modelů (LLM) ke zefektivnění a optimalizaci procesu.

Destilace promptů je vícestupňová technika, která zahrnuje využití LLM k asistenci při tvorbě, vylepšování a optimalizaci promptů. Místo spoléhání se pouze na lidskou expertízu a intuici tento přístup využívá znalosti a generativní schopnosti LLM k společnému vytváření vysoce kvalitních promptů.

Zapojením do iterativního procesu generování, vylepšování a integrace vám Destilace promptů umožňuje vytvářet prompty, které jsou koherentnější, komplexnější a lépe sladěné s požadovaným úkolem nebo výstupem. Všimněte si, že proces destilace lze provádět manuálně v jednom z mnoha "playgroundů" poskytovaných velkými AI společnostmi jako OpenAI nebo Anthropic, nebo může být automatizován jako součást kódu vaší aplikace, v závislosti na případu použití.

Jak to funguje

Destilace promptů typicky zahrnuje následující kroky:

1. **Identifikace hlavního záměru**: Analyzujte prompt k určení jeho primárního účelu a požadovaného výsledku. Odstraňte veškeré nadbytečné informace a zaměřte se na hlavní záměr promptu.
2. **Eliminace nejednoznačnosti**: Zkontrolujte prompt na přítomnost nejednoznačného nebo nejasného jazyka. Vyjasněte význam a poskytněte konkrétní detaily pro nasměrování AI k generování přesných a relevantních odpovědí.
3. **Zjednodušení jazyka**: Zjednodušte prompt použitím jasného a stručného jazyka. Vyhněte se složitým větným strukturám, žargonu nebo zbytečným detailům, které by mohly AI zmást nebo vnést šum.
4. **Poskytnutí relevantního kontextu**: Zahrňte pouze nejrelevantnější kontextové informace potřebné pro efektivní porozumění a zpracování promptu AI. Vyhněte

se zahrnutí irelevantních nebo redundantních detailů, které by mohly odvádět pozornost od hlavního záměru.

5. **Iterace a vylepšování**: Průběžně iterujte a vylepšujte prompt na základě odpovědí AI a zpětné vazby. Vyhodnocujte generované výstupy a provádějte nezbytné úpravy pro zlepšení srozumitelnosti a efektivity promptu. Volitelně můžete verzovat své prompty v databázi pomocí Prompt Object pro sledování iterací a možnost snadného vrácení změn za běhu.

Počáteční generování promptů

Prvním krokem v destilaci promptů je poskytnout LLM vysokoúrovňový popis nebo záměr požadovaného úkolu či výstupu. Může jít o stručné prohlášení nebo sadu klíčových slov, které vystihují podstatu toho, čeho má prompt dosáhnout. LLM pak na základě tohoto vstupu vygeneruje počáteční prompt, přičemž zahrne relevantní instrukce, příklady nebo kontext, které považuje za vhodné.

Pojďme zkusit dát Claude 3 Opus následující počáteční prompt:

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

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

Claude odpovídá následujícím výstupem:

```
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.
```

Vylepšování promptu

Jakmile je vytvořen počáteční prompt, zkontrolujte jej a poté požádejte VJM o zvážení jakéhokoli dodatečného kontextu, omezení či zpětné vazby. Nechte VJM vygenerovat vylepšenou verzi promptu, která bere v úvahu poskytnutý vstup a jeho vlastní porozumění úkolu. Tento proces vylepšování lze opakovat iterativně, přičemž každá iterace dále zdokonaluje prompt na základě spolupráce mezi uživatelem a VJM.

Zde je můj požadavek na vylepšení, který navazuje na předchozí proces destilace.

```
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.
```

A Claudova odpověď:

```
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.
```

Komprese promptu

Jak jste si možná všimli, LLM mají tendenci být mnohomluvné, pokud je nepožádáte o stručnost. Prompt, na kterém jsme pracovali v předchozích částech, je poměrně dlouhý, pojďme tedy požádat Clauda, aby ho zestručnil:

```
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 odpovídá s následujícím výrazně kratším promptem:

```
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.
```

Systémová direktiva a integrace kontextu

Kromě vylepšování samotného promptu může LLM také generovat vhodné systémové direktivy nebo kontextové informace pro usměrnění výsledného výstupu. Při prompt engineeringu AI rutin, které budou integrovány do vašeho aplikačního kódu, se v této fázi destilace budete téměř jistě soustředit na výstupní omezení, ale můžete také pracovat na požadovaném tónu, stylu, formátu nebo jakýchkoli jiných relevantních parametrech, které ovlivňují generovanou odpověď.

Finální sestavení promptu

Vyvrcholením procesu destilace promptů je sestavení finálního promptu. To zahrnuje kombinaci vylepšeného promptu, vygenerovaných systémových direktiv a integrovaného kontextu do souvislého a komplexního kódu, který je připraven k použití pro generování požadovaného výstupu.

Ve fázi finálního sestavení promptu můžete znovu experimentovat s kompresí promptu tím, že požádáte LLM o zkrácení formulace promptu na nejkratší možnou sérii tokenů při zachování podstaty jeho chování. Je to určitě metoda pokus-omyl, ale zejména v případě promptů, které budou spouštěny ve velkém měřítku, vám mohou úspory efektivity ušetřit značné množství peněz při spotřebě tokenů.

Klíčové výhody

Využitím znalostí a generativních schopností LLM k vylepšení vašich promptů je pravděpodobnější, že výsledné prompty budou dobře strukturované, informativní a přizpůsobené konkrétnímu úkolu. Iterativní proces vylepšování pomáhá zajistit, že prompty jsou kvalitní a efektivně zachycují požadovaný záměr. Mezi další výhody patří:

Efektivita a rychlost: Destilace promptů zefektivňuje proces prompt engineeringu automatizací určitých aspektů tvorby a vylepšování promptů. Kolaborativní povaha této techniky umožňuje rychlejší konvergenci k efektivnímu promptu, čímž snižuje čas a úsilí potřebné pro manuální tvorbu promptů.

Konzistence a škálovatelnost: Použití LLM v procesu prompt engineeringu pomáhá udržovat konzistenci napříč prompty, protože LLM se mohou učit a aplikovat osvědčené postupy a vzory z předchozích úspěšných promptů. Tato konzistence spolu se schopností generovat prompty ve velkém měřítku činí z destilace promptů cennou techniku pro rozsáhlé aplikace využívající umělou inteligenci.

Nápad na projekt: Nástroje na úrovni knihovny, které zjednodušují proces verzování promptů a hodnocení v systémech, které provádějí automatizované destilace promptů jako součást svého aplikačního kódu.

Pro implementaci destilace promptů mohou vývojáři navrhnout workflow nebo pipeline, která integruje LLM v různých fázích procesu prompt engineeringu. Toho

lze dosáhnout prostřednictvím API volání, vlastních nástrojů nebo integrovaných vývojových prostředí, která usnadňují plynulou interakci mezi uživateli a LLM během tvorby promptů. Konkrétní implementační detaily se mohou lišit v závislosti na zvoleném LLM platformě a požadavcích aplikace.

Co fine-tuning?

V této knize se podrobně věnujeme prompt engineeringu a RAG, ale ne fine-tuningu. Hlavním důvodem tohoto rozhodnutí je, že podle mého názoru většina vývojářů aplikací nepotřebuje fine-tuning pro své potřeby integrace AI.

Prompt engineering, který zahrnuje pečlivé vytváření promptů s nulovým až minimálním počtem ukázek, omezeními a instrukcemi, může efektivně navést model ke generování relevantních a přesných odpovědí pro širokou škálu úkolů. Poskytnutím jasného kontextu a zúžením cesty pomocí dobře navržených promptů můžete využít rozsáhlé znalosti velkých jazykových modelů bez potřeby fine-tuningu.

Podobně Generování s rozšířeným vyhledáváním (RAG) nabízí výkonný přístup k integraci AI do aplikací. Dynamickým získáváním relevantních informací z externích znalostních bází nebo dokumentů poskytuje RAG modelu zaměřený kontext v době promptování. To umožňuje modelu generovat odpovědi, které jsou přesnější, aktuálnější a specifičtější pro danou doménu, bez nutnosti časově a zdrojově náročného procesu fine-tuningu.

Zatímco fine--tuning může být přínosný pro vysoce specializované domény nebo úkoly vyžadující hlubokou úroveň přizpůsobení, často s sebou přináší významné výpočetní náklady, požadavky na data a režii údržby. Pro většinu scénářů vývoje aplikací by měla kombinace efektivního prompt engineeringu a RAG stačit k dosažení požadované funkcionality a uživatelské zkušenosti založené na AI.

Generování rozšířené o vyhledávání (RAG)

Co je Generování rozšířené o vyhledávání?

Jak RAG funguje?

Proč používat RAG ve vašich aplikacích?

Implementace RAG ve vaší aplikaci

Příprava zdrojů znalostí (Chunking)

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Rozdělení na propozice

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Poznámky k implementaci

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Kontrola kvality

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Výhody vyhledávání založeného na propozicích

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Příklady RAG v praxi

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Případová studie: RAG v aplikaci pro přípravu daní bez využití embeddingů

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Inteligentní optimalizace dotazů (IQO)

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Přeřazování

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Hodnocení RAG (RAGAs)

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Věrnost

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Relevance odpovědi

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Přesnost kontextu

Relevance kontextu

Úplnost kontextu

Úplnost entit kontextu

Sémantická podobnost odpovědí (ANSS)

Správnost odpovědi

Aspektová kritika

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Výzvy a budoucí výhled

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Sémantická segmentace: Vylepšení vyhledávání pomocí kontextově vědomé segmentace

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Hierarchické indexování: Strukturování dat pro lepší vyhledávání

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Self-RAG: Seberefexivní vylepšení

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

HyDE: Hypotetické dokumentové vnoření

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Co je kontrastní učení?

Množství pracovníků

Rád přemýšlím o svých AI komponentách jako o malých, téměř lidských virtuálních “pracovnících”, které lze bezproblémově integrovat do logiky mé aplikace pro plnění specifických úkolů nebo složitých rozhodnutí. Smyslem je záměrně polidštit schopnosti LLM, aby se nikdo příliš nenadchl a nepřiřazoval jim magické vlastnosti, které nemají.

Místo spoléhání se výhradně na složité algoritmy nebo časově náročné manuální implementace mohou vývojáři pojmout AI komponenty jako inteligentní, oddané, lidsky působící entity, které lze kdykoli vyvolat k řešení komplexních problémů a poskytování řešení založených na jejich tréninku a znalostech. Tyto entity se nenechají rozptýlit ani neonemocní. Spontánně se nerozhodnou dělat věci jinak, než jak jim bylo zadáno, a obecně řečeno, pokud jsou správně naprogramovány, nedělají ani chyby.

Z technického hlediska je klíčovým principem tohoto přístupu rozklad složitých úkolů nebo rozhodovacích procesů na menší, lépe zvládnutelné jednotky, které mohou být

zpracovány specializovanými AI pracovníky. Každý pracovník je navržen tak, aby se soustředil na konkrétní aspekt problému a přinášel své jedinečné odborné znalosti a schopnosti. Rozdělením pracovní zátěže mezi více AI pracovníků může aplikace dosáhnout větší efektivity, škálovatelnosti a přizpůsobivosti.

Vezměme si například webovou aplikaci, která vyžaduje moderování uživatelsky generovaného obsahu v reálném čase. Implementace komplexního moderačního systému od základu by byl náročný úkol vyžadující významné vývojářské úsilí a průběžnou údržbu. Využitím přístupu Množství pracovníků však mohou vývojáři integrovat AI moderační pracovníky do logiky aplikace. Tito pracovníci mohou automaticky analyzovat a označovat nevhodný obsah, což vývojářům umožní soustředit se na další kritické aspekty aplikace.

AI pracovníci jako nezávislé znovupoužitelné komponenty

Klíčovým aspektem přístupu Množství pracovníků je jeho modularita. Zastánci objektově orientovaného programování nám už desetiletí říkají, abychom o interakcích objektů přemýšleli jako o zprávách. AI pracovníci mohou být navrženi jako nezávislé, znovupoužitelné komponenty, které spolu mohou "mluvit" prostřednictvím zpráv v přirozeném jazyce, téměř jako kdyby to byli skuteční malí lidé mluvící spolu. Tento volně propojený přístup umožňuje aplikaci se v průběhu času přizpůsobovat a vyvíjet, jak se objevují nové AI technologie nebo se mění požadavky obchodní logiky.

V praxi se potřeba navrhovat jasná rozhraní a komunikační protokoly mezi komponentami nezměnila jen proto, že jsou zapojeni AI pracovníci. Stále musíte brát v úvahu i další faktory jako výkon, škálovatelnost a bezpečnost, ale nyní je třeba zvážit i zcela nové "měkké požadavky". Například mnoho uživatelů nesouhlasí s tím, aby jejich soukromá data byla použita k trénování nových AI modelů. Ověřili jste úroveň soukromí poskytovanou poskytovatelem modelu, který používáte?

AI pracovníci jako mikroslužby?

Při čtení o přístupu Množství pracovníků si možná všimnete určitých podobností s architekturou mikroslužeb. Oba přístupy zdůrazňují rozklad komplexních systémů na menší, lépe zvládnutelné a nezávisle nasaditelné jednotky. Stejně jako jsou mikroslužby navrženy tak, aby byly volně propojené, zaměřené na specifické obchodní schopnosti a komunikovaly prostřednictvím dobře definovaných API, jsou AI pracovníci navrženi tak, aby byli modulární, specializovaní na své úkoly a vzájemně interagovali prostřednictvím jasných rozhraní a komunikačních protokolů.

Existují však některé klíčové rozdíly, které je třeba mít na paměti. Zatímco mikroslužby jsou typicky implementovány jako samostatné procesy nebo služby běžící na různých strojích nebo kontejnerech, AI pracovníci mohou být implementováni jako samostatné komponenty v rámci jediné aplikace nebo jako samostatné služby, v závislosti na vašich specifických požadavcích a potřebách škálovatelnosti. Kromě toho komunikace mezi AI pracovníky často zahrnuje výměnu bohatých informací založených na přirozeném jazyce, jako jsou pokyny, instrukce a generovaný obsah, spíše než strukturovanější datové formáty běžně používané v mikroslužbách.

Navzdory těmto rozdílům zůstávají principy modularity, volného propojení a jasných komunikačních rozhraní ústředními pro oba vzory. Aplikováním těchto principů na vaši architekturu AI pracovníků můžete vytvářet flexibilní, škálovatelné a udržovatelné systémy, které využívají sílu AI k řešení komplexních problémů a poskytování hodnoty vašim uživatelům.

Přístup Množství pracovníků lze aplikovat v různých doménách a aplikacích, využívající sílu AI k řešení komplexních úkolů a poskytování inteligentních řešení. Podívejme se na

několik konkrétních příkladů, jak lze AI pracovníky využít v různých kontextech.

Správa účtů

Prakticky každá samostatná webová aplikace má koncept účtu (nebo uživatele). V Olympii používáme AI pracovníka `AccountManager`, který je naprogramován tak, aby dokázal zpracovat různé druhy požadavků na změny související s uživatelskými účty.

Jeho direktiva vypadá takto:

```
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.
```

Počáteční stav účtu vytvořený pomocí `account.to_directive` je jednoduše textový popis účtu, včetně relevantních souvisejících dat, jako jsou uživatelé, předplatné atd.

Rozsah funkcí dostupných pro `AccountManager` mu dává možnost upravovat předplatné uživatele, přidávat a odebírat AI konzultanty a další druhy placených doplňků a zasílat notifikační e-maily vlastníkovi účtu. Kromě funkce `finished` může také `notify_human_administrator` v případě, že během zpracování narazí na chybu nebo potřebuje jakýkoli jiný druh asistence s požadavkem.

Všimněte si, že v případě dotazů se může `AccountManager` rozhodnout prohledat znalostní bázi Olympie, kde najde instrukce, jak zacházet s hraničními případy a jakoukoliv jinou situací, ve které si není jistý, jak postupovat.

Využití v e-commerce

V oblasti e-commerce mohou AI pracovníci hrát klíčovou roli při vylepšování uživatelské zkušenosti a optimalizaci obchodních operací. Zde je několik způsobů, jak lze AI pracovníky využít:

Produktová doporučení

Jednou z nejsilnějších aplikací AI pracovníků v e-commerce je generování personalizovaných produktových doporučení. Analyzováním chování uživatelů, historie nákupů a preferencí mohou tito pracovníci navrhovat produkty, které jsou přizpůsobené zájmům a potřebám každého jednotlivého uživatele.

Klíčem k efektivním produktovým doporučením je využití kombinace kolaborativního filtrování a filtrování založeného na obsahu. Kolaborativní filtrování sleduje chování podobných uživatelů k identifikaci vzorců a vytváření doporučení na základě toho, co nakoupili nebo co se líbilo ostatním s podobnými preferencemi. Filtrování založené na obsahu se naopak zaměřuje na charakteristiky a atributy samotných produktů a doporučuje položky, které sdílejí podobné vlastnosti s těmi, o které uživatel dříve projevil zájem.

Zde je zjednodušený příklad implementace pracovníka pro doporučování produktů v Ruby, tentokrát s využitím “Railway Oriented (ROP)” funkcionálního stylu programování:

```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
```

Styl funkcionálního programování v Ruby použitý v příkladu je ovlivněn jazyky F# a Rust. Více se o něm můžete dočíst v vysvětlení této techniky od mého přítele Chada Wooleye na GitLabu.

V tomto příkladu `ProductRecommendationWorker` přijímá uživatele jako vstup a generuje personalizovaná doporučení produktů předáváním hodnotového objektu skrze řetězec funkcionálních kroků. Pojďme si rozebrat každý krok:

1. `ValidateUser.validate`: Tento krok zajišťuje, že je uživatel platný a způsobilý pro personalizovaná doporučení. Kontroluje, zda uživatel existuje, je aktivní a má k dispozici potřebná data pro generování doporučení. Pokud validace selže, je vrácen chybový výsledek a řetězec je přerušen.
2. `AnalyzeCurrentSession.analyze`: Pokud je uživatel platný, tento krok analyzuje aktuální relaci prohlížení uživatele pro získání kontextuálních

informací. Sleduje nedávné interakce uživatele, jako jsou zobrazené produkty, vyhledávací dotazy a obsah košíku, aby pochopil jejich současné zájmy a záměry.

3. `CollaborativeFilter.filter`: S využitím *chování podobných uživatelů* tento krok aplikuje techniky kolaborativního filtrování k identifikaci produktů, které by mohly uživatele zajímat. Bere v úvahu faktory jako historie nákupů, hodnocení a interakce uživatelů s položkami pro vytvoření sady kandidátních doporučení.
4. `ContentBasedFilter.filter`: Tento krok dále zpřesňuje kandidátní doporučení aplikací filtrování založeného na obsahu. Porovnává atributy a charakteristiky kandidátních produktů s *preferencemi uživatele a historickými daty* pro výběr nejrelevantnějších položek.
5. `ProductSelector.select`: Nakonec tento krok vybere N nejlepších produktů z filtrovaných doporučení na základě předem definovaných kritérií, jako je skóre relevance, popularita nebo další obchodní pravidla. Vybrané produkty jsou pak vráceny jako konečná personalizovaná doporučení.

Krása použití funkcionálního programovacího stylu v Ruby zde spočívá v tom, že nám umožňuje zřetězit tyto kroky jasným a stručným způsobem. Každý krok se zaměřuje na konkrétní úkol a vrací objekt `Result`, který může být buď úspěch (`ok`) nebo chyba (`err`). Pokud kterýkoli krok narazí na chybu, řetězec je přerušen a chyba je propagována do konečného výsledku.

V `case` příkazu na konci provádíme pattern matching konečného výsledku. Pokud je výsledkem chyba (`ProductRecommendationError`), zaznamenáme ji pomocí nástroje jako je Honeybadger pro účely monitorování a ladění. Pokud je výsledek úspěšný (`ProductRecommendations`), vysíláme událost `:new_recommendations` pomocí pub/sub knihovny Wisper, předávající uživatele a vygenerovaná doporučení.

Využitím technik funkcionálního programování můžeme vytvořit modulární a udržovatelný worker pro doporučování produktů. Každý krok je samostatný a lze jej snadno testovat, upravovat nebo nahradit bez ovlivnění celkového toku.

Použití pattern matchingu a třídy `Result` nám pomáhá elegantně zpracovávat chyby a zajišťuje, že worker selže rychle, pokud kterýkoli krok narazí na problém.

Samozřejmě se jedná o zjednodušený příklad a v reálném scénáři byste potřebovali integraci s vaší e-commerce platformou, zpracování krajních případů a dokonce se zabývat implementací doporučovacích algoritmů. Nicméně základní principy rozložení problému na menší kroky a využití technik funkcionálního programování zůstávají stejné.

Detekce podvodů

Zde je zjednodušený příklad implementace workeru pro detekci podvodů pomocí stejného stylu Railway Oriented Programming (ROP) v 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
```

Třída `FraudDetection` je *value object*, který zapouzdřuje stav detekce podvodů pro danou transakci. Poskytuje strukturovaný způsob analýzy a vyhodnocení rizika podvodu spojeného s transakcí na základě různých rizikových faktorů.

```
class FraudDetection
  RISK_THRESHOLD = 0.8

  attr_accessor :transaction, :risk_factors

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

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

  def high_risk?
    fraud_probability > RISK_THRESHOLD
  end

  private

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

Třída `FraudDetection` má následující atributy:

- `transaction`: Reference na transakci, která je analyzována z hlediska podvodu.
- `risk_factors`: Pole, které uchovává rizikové faktory spojené s transakcí. Každý rizikový faktor je reprezentován jako hash, kde klíč je popis rizikového faktoru a hodnota je pravděpodobnost podvodu spojená s tímto rizikovým faktorem.

Metoda `add_risk_factor` umožňuje přidání rizikového faktoru do pole `risk_factors`. Přijímá dva parametry: `description`, což je řetězec popisující rizikový faktor, a `probability`, což je float reprezentující pravděpodobnost podvodu spojenou s tímto rizikovým faktorem. Pro jednoduchou kontrolu typů používáme podmínku `case..in`.

Metoda `high_risk?`, která bude kontrolována na konci řetězce, je predikátová metoda, která porovnává `fraud_probability` (vypočítanou součtem pravděpodobností všech rizikových faktorů) s hodnotou `RISK_THRESHOLD`.

Třída `FraudDetection` poskytuje čistý a zapouzdřený způsob správy detekce podvodů pro transakci. Umožňuje přidávat více rizikových faktorů, každý s vlastním popisem a pravděpodobností, a poskytuje metodu pro určení, zda je transakce považována za vysoce rizikovou na základě vypočítané pravděpodobnosti podvodu. Třídu lze snadno integrovat do většího systému detekce podvodů, kde různé komponenty mohou spolupracovat při hodnocení a zmírňování rizika podvodných transakcí.

A konečně, protože toto je přece jen kniha o programování s využitím AI, zde je ukázka implementace třídy `CheckCustomerHistory` využívající AI zpracování pomocí modulu `ChatCompletion` z mé knihovny Raix:

```
1  class CheckCustomerHistory
2    include Raix::ChatCompletion
3
4    attr_accessor :fraud_detection
5
6    INSTRUCTION = <<~END
7      You are an AI assistant tasked with checking a customer's transaction
8      history for potential fraud indicators. Given the current transaction
9      and the customer's past transactions, analyze the data to identify any
10     suspicious patterns or anomalies.
11
12     Consider factors such as the frequency of transactions, transaction
13     amounts, geographical locations, and any deviations from the customer's
14     typical behavior to generate a probability score as a float in the range
15     of 0 to 1 (with 1 being absolute certainty of fraud).
16
```

```
    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
    tx_history = fraud_detection.transaction.user.tx_history
    [
      { system: INSTRUCTION },
      { user: "Transaction history: #{tx_history.to_json}" },
      { assistant: "OK. Please provide the current transaction." },
      { user: "Current transaction: #{fraud_detection.transaction.to_json}" }
    ]
  end
end
```

V tomto příkladu `CheckCustomerHistory` definuje konstantu `INSTRUCTION`, která poskytuje modelu umělé inteligence konkrétní pokyny k analýze historie transakcí zákazníka pro potenciální indikátory podvodů prostřednictvím systémové direktivy.

Metoda `self.check` je třídní metoda, která inicializuje novou instanci

`CheckCustomerHistory` s objektem `fraud_detection` a volá metodu `call` k provedení analýzy historie zákazníka.

Uvnitř metody `call` je získána historie transakcí zákazníka a zformátována do přepisu, který je předán modelu umělé inteligence. Model umělé inteligence analyzuje historii transakcí na základě poskytnutých instrukcí a vrací souhrn svých zjištění.

Zjištění jsou přidána do objektu `fraud_detection` a aktualizovaný objekt `fraud_detection` je vrácen jako úspěšný `Result`.

Využitím modulu `ChatCompletion` může třída `CheckCustomerHistory` využít sílu umělé inteligence k analýze historie transakcí zákazníka a identifikaci potenciálních indikátorů podvodů. To umožňuje sofistikovanější a adaptivnější techniky detekce podvodů, protože model umělé inteligence se může učit a přizpůsobovat novým vzorům a anomáliím v průběhu času.

Aktualizovaný `FraudDetectionWorker` a třída `CheckCustomerHistory` demonstrují, jak lze bezproblémově integrovat AI workery, čímž se vylepšuje proces detekce podvodů o schopnosti inteligentní analýzy a rozhodování.

Analýza sentimentu zákazníků

Zde je ještě jeden podobný příklad, jak můžete implementovat workera pro analýzu sentimentu zákazníků. Tentokrát s mnohem méně vysvětlování, protože byste již měli chápat, jak tento styl programování funguje:

```
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
```

V tomto příkladu kroky `CustomerSentimentAnalysisWorker` zahrnují předzpracování zpětné vazby (např. odstranění šumu, tokenizaci), provedení analýzy sentimentu pro určení celkového sentimentu (pozitivní, negativní nebo neutrální), extrakci klíčových frází a témat, identifikaci trendů a vzorců a generování využitelných poznatků na základě analýzy.

Aplikace ve zdravotnictví

V oblasti zdravotnictví mohou AI pracovníci pomáhat zdravotnickým odborníkům a výzkumníkům v různých úkolech, což vede ke zlepšení výsledků pacientů a urychlení lékařských objevů. Některé příklady zahrnují:

Příjem pacientů

AI pracovníci mohou zefektivnit proces příjmu pacientů automatizací různých úkolů a poskytováním inteligentní asistence.

Plánování schůzek: AI pracovníci mohou spravovat plánování schůzek tím, že rozumí preferencím pacientů, jejich dostupnosti a naléhavosti jejich zdravotních potřeb. Mohou komunikovat s pacienty prostřednictvím konverzačních rozhraní, provázet je procesem plánování a najít nejvhodnější termíny schůzek na základě požadavků pacienta a dostupnosti poskytovatele zdravotní péče.

Sběr zdravotní anamnézy: Během příjmu pacientů mohou AI pracovníci pomáhat při shromažďování a dokumentaci zdravotní anamnézy pacienta. Mohou vést interaktivní dialogy s pacienty, klást relevantní otázky o jejich předchozích zdravotních stavech, lécích, alergiích a rodinné anamnéze. AI pracovníci mohou využívat techniky zpracování přirozeného jazyka k interpretaci a strukturování shromážděných informací, zajišťujíc jejich přesné zachycení v elektronické zdravotní dokumentaci pacienta.

Hodnocení a stratifikace příznaků: AI pracovníci mohou provádět úvodní hodnocení příznaků tím, že se ptají pacientů na jejich současné příznaky, trvání, závažnost a související faktory. Využitím lékařských znalostních bází a modelů strojového učení mohou tito pracovníci analyzovat poskytnuté informace a generovat předběžné diferenciální diagnózy nebo doporučovat vhodné další kroky, jako je naplánování konzultace se zdravotnickým pracovníkem nebo navržení opatření pro samostatnou péči.

Ověření pojištění: AI pracovníci mohou pomáhat s ověřováním pojištění během příjmu pacientů. Mohou shromažďovat údaje o pojištění pacientů, komunikovat s pojišťovnami prostřednictvím API nebo webových služeb a ověřovat způsobilost k pojištění a výhody. Tato automatizace pomáhá zefektivnit proces ověřování pojištění, snižuje administrativní zátěž a zajišťuje přesné zachycení informací.

Vzdělávání pacientů a pokyny: AI pracovníci mohou poskytovat pacientům relevantní

vzdělávací materiály a pokyny na základě jejich specifických zdravotních stavů nebo nadcházejících procedur. Mohou dodávat personalizovaný obsah, odpovídat na běžné otázky a poskytovat pokyny k přípravě před návštěvou, instrukcím k užívání léků nebo následné péči. To pomáhá udržovat pacienty informované a zapojené během jejich zdravotní cesty.

Využitím AI pracovníků při příjmu pacientů mohou zdravotnické organizace zvýšit efektivitu, snížit čekací doby a zlepšit celkovou zkušenost pacientů. Tito pracovníci mohou zvládat rutinní úkoly, shromažďovat přesné informace a poskytovat personalizovanou asistenci, což umožňuje zdravotnickým pracovníkům soustředit se na poskytování vysoce kvalitní péče pacientům.

Hodnocení rizik pacientů

AI pracovníci mohou hrát klíčovou roli při hodnocení rizik pacientů analýzou různých zdrojů dat a aplikací pokročilých analytických technik.

Integrace dat: AI pracovníci mohou shromažďovat a zpracovávat data pacientů z různých zdrojů, jako je elektronická zdravotní dokumentace, lékařské zobrazování, laboratorní výsledky, nositelná zařízení a sociální determinanty zdraví. Konsolidací těchto informací do komplexního profilu pacienta mohou AI pracovníci poskytnout holistický pohled na zdravotní stav pacienta a rizikové faktory.

Stratifikace rizik: AI pracovníci mohou používat prediktivní modely ke stratifikaci pacientů do různých rizikových kategorií na základě jejich individuálních charakteristik a zdravotních dat. Tato stratifikace rizik umožňuje poskytovatelům zdravotní péče prioritizovat pacienty, kteří vyžadují bezprostřednější pozornost nebo intervenci. Například pacienti identifikovaní jako vysoce rizikoví pro určitý stav mohou být označeni pro bližší sledování, preventivní opatření nebo včasnou intervenci.

Personalizované rizikové profily: AI pracovníci mohou generovat personalizované rizikové profily pro každého pacienta, zdůrazňující specifické faktory přispívající

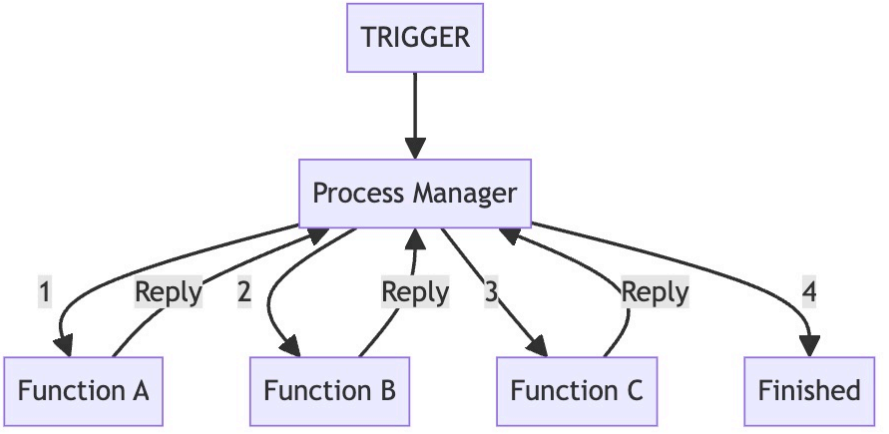
k jejich rizikovým skóre. Tyto profily mohou zahrnovat poznatky o životním stylu pacienta, genetických predispozicích, environmentálních faktorech a sociálních determinantech zdraví. Poskytnutím detailního rozkladu rizikových faktorů mohou AI pracovníci pomoci poskytovatelům zdravotní péče přizpůsobit strategie prevence a léčebné plány individuálním potřebám pacientů.

Kontinuální monitoring rizik: AI pracovníci mohou nepřetržitě sledovat data pacientů a aktualizovat hodnocení rizik v reálném čase. Když se objeví nové informace, jako jsou změny životních funkcí, laboratorních výsledků nebo dodržování léčby, AI pracovníci mohou přepočítat riziková skóre a upozornit poskytovatele zdravotní péče na významné změny. Toto proaktivní sledování umožňuje včasné intervence a úpravy plánů péče o pacienty.

Podpora klinického rozhodování: AI pracovníci mohou integrovat výsledky hodnocení rizik do systémů podpory klinického rozhodování, poskytujíc poskytovatelům zdravotní péče doporučení a upozornění založená na důkazech. Například pokud rizikové skóre pacienta pro určitý stav překročí určitou hranici, může AI pracovník upozornit poskytovatele zdravotní péče, aby zvážil specifické diagnostické testy, preventivní opatření nebo možnosti léčby na základě klinických směrnic a osvědčených postupů.

Tito pracovníci dokážou zpracovávat obrovské množství pacientských dat, aplikovat sofistikované analýzy a generovat využitelné poznatky pro podporu klinického rozhodování. To v konečném důsledku vede ke zlepšení výsledků pacientů, snížení nákladů na zdravotní péči a lepšímu řízení zdraví populace.

AI pracovník jako správce procesů



V kontextu aplikací řízených umělou inteligencí může být pracovník navržen tak, aby fungoval jako Správce procesů, jak je popsáno v knize "Enterprise Integration Patterns" od Gregora Hohpeho. Správce procesů je centrální komponenta, která udržuje stav procesu a určuje další kroky zpracování na základě průběžných výsledků.

Když AI pracovník působí jako Správce procesů, přijme příchozí zprávu, která inicializuje proces, známou jako *spouštěcí zpráva*. AI pracovník pak udržuje stav provádění procesu (jako přepis konverzace) a zpracovává zprávu prostřednictvím série kroků zpracování implementovaných jako nástrojové funkce, které mohou být sekvenční nebo paralelní a jsou volány podle jeho uvážení.

Pokud používáte třídu AI modelů jako GPT--4, která umí spouštět funkce paralelně, může váš pracovník provádět více kroků současně. Přiznávám, že jsem to sám nezkoušel a můj instinkt říká, že výsledky se mohou lišit.

Po každém jednotlivém kroku zpracování se řízení vrátí zpět k AI pracovníkovi, což

mu umožňuje určit další krok(y) zpracování na základě aktuálního stavu a získaných výsledků.

Ukládejte své spouštěcí zprávy

Ze své zkušenosti mohu říct, že je rozumné implementovat spouštěcí zprávu jako objekt založený na databázi. Tímto způsobem je každá instance procesu identifikována jedinečným primárním klíčem a poskytuje místo pro uložení stavu spojeného s prováděním, včetně přepisu konverzace s AI.

Zde je například zjednodušená verze třídy modelu `AccountChange` z Olympie, která představuje požadavek na provedení změny v uživatelském účtu.

```ruby
# == 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

```

```
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
```

Třída `AccountChange` slouží jako spouštěcí zpráva, která zahajuje proces zpracování požadavku na změnu účtu. Všimněte si, jak je vysílána do subsystému pub/sub systému Olympia založeném na Wisper po dokončení transakce vytvoření.

Ukládání spouštěcí zprávy do databáze tímto způsobem poskytuje trvalý záznam o každém požadavku na změnu účtu. Každé instanci třídy `AccountChange` je přiřazen jedinečný primární klíč, což umožňuje snadnou identifikaci a sledování jednotlivých požadavků. To je obzvláště užitečné pro účely auditního logování, protože to systému umožňuje udržovat historický záznam všech změn účtu, včetně toho, kdy byly požadovány, jaké změny byly požadovány a jaký je aktuální stav každého požadavku.

V uvedeném příkladu třída `AccountChange` obsahuje pole jako `description` pro zachycení podrobností požadované změny, `state` pro reprezentaci aktuálního stavu požadavku (např. requested, complete, requires_human_review) a `transcript` pro uložení přepisu konverzace s AI související s požadavkem. Pole `description` je skutečný prompt, který se používá k zahájení první chat completion s AI. Ukládání těchto dat poskytuje cenný kontext a umožňuje lepší sledování a analýzu procesu změny účtu.

Ukládání spouštěcích zpráv v databázi umožňuje robustní zpracování chyb a zotavení. Pokud během zpracování požadavku na změnu účtu dojde k chybě, systém označí

požadavek jako neúspěšný a převede jej do stavu, který vyžaduje lidský zásah. Tím je zajištěno, že žádný požadavek není ztracen ani zapomenut a všechny problémy mohou být řádně řešeny.

AI worker, jako Správce procesů, poskytuje centrální bod kontroly a umožňuje výkonné možnosti reportování a ladění procesů. Je však důležité poznamenat, že použití AI workera jako Správce procesů pro každý pracovní scénář ve vaší aplikaci může být přehnané.

Integrace AI Workers do architektury vaší aplikace

Při začleňování AI workers do architektury vaší aplikace je třeba řešit několik technických aspektů, aby byla zajištěna plynulá integrace a efektivní komunikace mezi AI workers a ostatními komponenty aplikace. Tato část se zabývá klíčovými aspekty navrhování těchto rozhraní, zpracování toku dat a správy životního cyklu AI workers.

Navrhování jasných rozhraní a komunikačních protokolů

Pro usnadnění bezproblémové integrace mezi AI workers a ostatními komponenty aplikace je zásadní definovat jasná rozhraní a komunikační protokoly. Zvažte následující přístupy:

Integrace založená na API: Vystavte funkcionalitu AI workers prostřednictvím dobře definovaných API, jako jsou RESTful endpointy nebo GraphQL schémata. To umožňuje ostatním komponentám komunikovat s AI workers pomocí standardních HTTP požadavků a odpovědí. Integrace založená na API poskytuje jasnou smlouvu mezi AI workers a konzumujícími komponentami, což usnadňuje vývoj, testování a údržbu integračních bodů.

Komunikace založená na zprávách: Implementujte vzory komunikace založené na zprávách, jako jsou fronty zpráv nebo systémy typu publisher--subscriber, které umožňují asynchronní interakci mezi AI workers a ostatními komponentami. Tento přístup odděluje AI workers od zbytku aplikace, což umožňuje lepší škálovatelnost, odolnost vůči chybám a volné propojení. Komunikace založená na zprávách je obzvláště užitečná, když je zpracování prováděné AI workers časově náročné nebo náročné na zdroje, protože umožňuje ostatním částem aplikace pokračovat v provádění bez čekání na dokončení úkolů AI workers.

Událostmi řízená architektura: Navrhněte svůj systém kolem událostí a spouštěčů, které aktivují AI workers, když jsou splněny specifické podmínky. AI workers se mohou přihlásit k odběru relevantních událostí a podle toho reagovat, vykonávat své určené úkoly, když události nastanou. Událostmi řízená architektura umožňuje zpracování v reálném čase a umožňuje vyvolávat AI workers na vyžádání, což snižuje zbytečnou spotřebu zdrojů. Tento přístup je vhodný pro scénáře, kde AI workers musí reagovat na konkrétní akce nebo změny ve stavu aplikace.

Zpracování toku dat a synchronizace

Při integraci AI workers do vaší aplikace je zásadní zajistit plynulý tok dat a udržovat konzistenci dat mezi AI workers a ostatními komponentami. Zvažte následující aspekty:

Příprava dat: Před vložením dat do AI workers možná budete muset provést různé úkoly přípravy dat, jako je čištění, formátování a/nebo transformace vstupních dat. Nejen že chcete zajistit, aby AI workers mohli efektivně zpracovávat, ale také chcete zajistit, že neplýtváte tokeny věnováním pozornosti informacím, které worker může považovat v nejlepším případě za zbytečné, v nejhorším případě za rušivé. Příprava dat může zahrnovat úkoly jako odstraňování šumu, zpracování chybějících hodnot nebo konverzi datových typů.

Perzistence dat: Jak budete ukládat a uchovávat data, která proudí do a z AI workers? Zvažte faktory jako objem dat, vzory dotazů a škálovatelnost. Potřebujete uchovávat

přepis AI jako reflexi jeho "myšlenkového procesu" pro účely auditu nebo ladění, nebo stačí mít záznam pouze o výsledcích?

Získávání dat: Získávání dat potřebných pro pracovníky může zahrnovat dotazování databází, čtení ze souborů nebo přístup k externím API. Ujistěte se, že zvážíte latenci a způsob, jakým budou mít AI pracovníci přístup k nejaktuálnějším datům. Potřebují plný přístup k vaší databázi, nebo byste měli úzce definovat rozsah jejich přístupu podle toho, co dělají? A co škálování? Zvažte mechanismy ukládání do mezipaměti pro zlepšení výkonu a snížení zátěže základních datových zdrojů.

Synchronizace dat: Když více komponent, včetně AI pracovníků, přistupuje k sdíleným datům a upravuje je, je důležité implementovat správné synchronizační mechanismy pro zachování konzistence dat. Strategie zamykání databází, jako je optimistické nebo pesimistické zamykání, vám mohou pomoci předcházet konfliktům a zajistit integritu dat. Implementujte techniky správy transakcí pro seskupení souvisejících datových operací a zachování vlastností ACID (atomicita, konzistence, izolace a trvalost)

Zpracování a zotavení z chyb: Implementujte robustní mechanismy pro zpracování chyb a zotavení, které se vypořádají s problémy souvisejícími s daty, jež mohou během procesu toku dat vzniknout. Elegantně zpracovávejte výjimky a poskytujte smysluplné chybové zprávy pro usnadnění ladění. Implementujte mechanismy opakování pokusů a záložní strategie pro řešení dočasných výpadků nebo přerušení sítě. Definujte jasné postupy pro obnovu dat v případě poškození nebo ztráty dat.

Pečlivým návrhem a implementací mechanismů toku a synchronizace dat můžete zajistit, že vaši AI pracovníci budou mít přístup k přesným, konzistentním a aktuálním datům. To jim umožní efektivně plnit své úkoly a produkovat spolehlivé výsledky.

Správa životního cyklu AI pracovníků

Vytvořte standardizovaný proces pro inicializaci a konfiguraci AI pracovníků. Osobně preferuji frameworky, které standardizují způsob definování nastavení, jako jsou názvy

modelů, systémové direktivy a definice funkcí. Zajistěte, aby byl proces inicializace automatizovaný a reprodukovatelný pro usnadnění nasazení a škálování.

Implementujte komplexní mechanismy monitorování a protokolování pro sledování stavu a výkonu AI pracovníků. Shromažďujte metriky jako využití zdrojů, doba zpracování, míra chybovosti a propustnost. Používejte centralizované logovací systémy jako ELK stack (Elasticsearch, Logstash, Kibana) pro agregaci a analýzu logů z více AI pracovníků.

Zabudujte odolnost proti chybám a pružnost do architektury AI pracovníků. Implementujte mechanismy pro zpracování chyb a zotavení, aby se elegantně vypořádaly s selháními nebo výjimkami. Velké jazykové modely jsou stále špičkovou technologií; poskytovatelé mají tendenci často nečekaně vypadávat. Používejte mechanismy opakování pokusů a jističe, abyste předešli kaskádovým selháním.

Kompozice a orchestrace AI pracovníků

Jednou z klíčových výhod architektury AI pracovníků je její komponovatelnost, která vám umožňuje kombinovat a orchestrovat více AI pracovníků pro řešení komplexních problémů. Rozdělením většího úkolu na menší, lépe zvládnutelné podúkoly, z nichž každý je zpracováván specializovaným AI pracovníkem, můžete vytvářet výkonné a flexibilní systémy. V této části prozkoumáme různé přístupy ke kompozici a orchestraci "množství" AI pracovníků.

Řetězení AI pracovníků pro vícekrokové pracovní postupy

V mnoha scénářích lze komplexní úkol rozložit na sérii postupných kroků, kde výstup jednoho AI pracovníka se stává vstupem pro dalšího. Toto řetězení AI pracovníků vytváří vícekrokový pracovní postup nebo pipeline. Každý AI pracovník v řetězci se zaměřuje na konkrétní podúkol a konečný výstup je výsledkem společného úsilí všech pracovníků.

Uvažujme příklad v kontextu aplikace Ruby on Rails pro zpracování uživatelsky generovaného obsahu. Pracovní postup zahrnuje následující kroky, které jsou přiznávě pravděpodobně každý příliš jednoduchý na to, aby stálo za to je v reálných případech takto rozkládat, ale usnadňují pochopení příkladu:

1. **Čištění textu**: AI pracovník zodpovědný za odstranění HTML tagů, převod textu na malá písmena a zpracování Unicode normalizace.

2. **Detekce jazyka**: AI pracovník, který identifikuje jazyk vyčištěného textu.

3. **Analýza sentimentu**: AI pracovník, který určuje sentiment (pozitivní, negativní nebo neutrální) textu na základě detekovaného jazyka.

4. **Kategorizace obsahu**: AI pracovník, který klasifikuje text do předdefinovaných kategorií pomocí technik zpracování přirozeného jazyka.

Zde je velmi zjednodušený příklad toho, jak můžete zřetězit tyto AI pracovníky pomocí 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
```

V tomto příkladu třída `ContentProcessor` inicializuje surový text a řetězí AI workery dohromady v metodě `process`. Každý AI worker provede svůj specifický úkol a předá výsledek dalšímu workeru v řetězci. Konečným výstupem je hash obsahující vyčištěný text, detekovaný jazyk, sentiment a kategorii obsahu.

Paralelní zpracování pro nezávislé AI workery

V předchozím příkladu jsou AI workery zřetězeny sekvenčně, kde každý worker zpracuje text a předá výsledek dalšímu workeru. Pokud však máte více AI workerů, které mohou pracovat nezávisle se stejným vstupem, můžete optimalizovat pracovní postup jejich paralelním zpracováním.

V daném scénáři, jakmile `TextCleanupWorker` provede čištění textu, mohou `LanguageDetectionWorker`, `SentimentAnalysisWorker` a `CategorizationWorker` všichni zpracovávat vyčištěný text nezávisle. Spuštěním těchto workerů paralelně můžete potenciálně snížit celkovou dobu zpracování a zlepšit efektivitu vašeho pracovního postupu.

Pro dosažení paralelního zpracování v Ruby můžete využít techniky souběžnosti, jako jsou vlákna nebo asynchronní programování. Zde je příklad, jak můžete upravit třídu `ContentProcessor` pro paralelní zpracování posledních tří workerů pomocí vláken:

```ruby
require 'concurrent'

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

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

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

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

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

```
    end

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

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

V této optimalizované verzi používáme knihovnu `concurrent-ruby` k vytvoření objektů `Concurrent::Future` pro každého z nezávislých AI workerů. `Future` představuje výpočet, který bude proveden asynchronně v samostatném vlákně.

Po kroku čištění textu vytvoříme tři objekty `Future`: `language_future`, `sentiment_future` a `category_future`. Každý `Future` spouští svého odpovídajícího AI workera (`LanguageDetectionWorker`, `SentimentAnalysisWorker` a `CategorizationWorker`) v samostatném vlákně, přičemž jako vstup předává `cleaned_text`.

Voláním metody `value` na každém `Future` čekáme na dokončení výpočtu a získáváme výsledek. Metoda `value` blokuje, dokud není výsledek k dispozici, čímž zajišťuje, že všichni paralelní workeři dokončili zpracování před pokračováním.

Nakonec sestavíme výstupní hash s vyčištěným textem a výsledky z paralelních workerů, stejně jako v původním příkladu.

Zpracováním nezávislých AI workerů paralelně můžete potenciálně snížit celkovou dobu zpracování ve srovnání se sekvenčním spouštěním. Tato optimalizace je obzvláště přínosná při práci s časově náročnými úlohami nebo při zpracování velkých objemů dat.

Je však důležité poznamenat, že skutečné výkonnostní zisky závisí na různých faktorech, jako je složitost každého workera, dostupné systémové prostředky a režie správy vláken. Je vždy dobrou praxí provádět měření výkonu a profilování kódu pro určení optimální úrovně paralelizace pro váš konkrétní případ použití.

Kromě toho při implementaci paralelního zpracování mějte na paměti všechny sdílené zdroje nebo závislosti mezi workery. Ujistěte se, že workeři mohou pracovat nezávisle bez konfliktů nebo souběžných podmínek. Pokud existují závislosti nebo sdílené zdroje, možná budete muset implementovat vhodné synchronizační mechanismy pro zachování integrity dat a vyhnout se problémům, jako jsou uváznutí nebo nekonzistentní výsledky.

Ruby's Global Interpreter Lock a asynchronní zpracování

Je důležité pochopit důsledky Global Interpreter Lock (GIL) v Ruby při zvažování asynchronního zpracování založeného na vláknech v Ruby.

GIL je mechanismus v interpreteru Ruby, který zajišťuje, že pouze jedno vlákno může v daný okamžik vykonávat Ruby kód, a to i na vícejadrových procesorech. To znamená, že zatímco v rámci Ruby procesu lze vytvořit a spravovat více vláken, pouze jedno vlákno může aktivně vykonávat Ruby kód v jakémkoli daném okamžiku.

GIL je navržen tak, aby zjednodušil implementaci Ruby interpreteru a poskytl bezpečnost vláken pro interní datové struktury Ruby. Nicméně také omezuje potenciál pro skutečně paralelní vykonávání Ruby kódu.

Když v Ruby používáte vlákna, například s knihovnou `concurrent-ruby` nebo vestavěnou třídou `Thread`, vlákna podléhají omezením GIL. GIL umožňuje každému vláknu vykonávat Ruby kód po krátký časový úsek před přepnutím na jiné vlákno, čímž vytváří iluzi souběžného vykonávání.

Nicméně kvůli GIL zůstává skutečné vykonávání Ruby kódu sekvenční. Zatímco jedno vlákno vykonává Ruby kód, ostatní vlákna jsou v podstatě pozastavena a čekají na svou řadu k získání GIL a vykonání.

To znamená, že asynchronní zpracování založené na vláknech v Ruby je nejefektivnější pro úlohy náročné na I/O, jako je čekání na odpovědi externích API

(například externě hostované velké jazykové modely) nebo provádění operací I/O se soubory. Když vlákno narazí na I/O operaci, může uvolnit GIL a umožnit ostatním vláknům vykonávat kód během čekání na dokončení I/O.

Na druhou stranu, pro úlohy náročné na CPU, jako jsou intenzivní výpočty nebo dlouhodobé zpracování AI workerů, může GIL omezit potenciální výkonnostní zisky paralelizmu založeného na vláknech. Protože pouze jedno vlákno může v daný okamžik vykonávat Ruby kód, celková doba vykonávání nemusí být významně snížena ve srovnání se sekvenčním zpracováním.

Pro dosažení skutečně paralelního vykonávání úloh náročných na CPU v Ruby možná budete muset prozkoumat alternativní přístupy, jako jsou:

- Použití paralelizmu založeného na procesech s více Ruby procesy, z nichž každý běží na samostatném jádru CPU.
- Využití externích knihoven nebo frameworků, které poskytují nativní rozšíření nebo rozhraní k jazykům bez GIL, jako jsou C nebo Rust.,
- Využití frameworků pro distribuované výpočty nebo front zpráv pro distribuci úloh mezi více počítači nebo procesy.

Při navrhování a implementaci asynchronního zpracování v Ruby je zásadní zvážit povahu vašich úloh a omezení daná GIL. Zatímco asynchronní zpracování založené na vláknech může přinést výhody pro úlohy náročné na I/O, nemusí nabídnout významná vylepšení výkonu pro úlohy náročné na CPU kvůli omezením GIL.

Ensemblové techniky pro zlepšení přesnosti

Ensemblové techniky zahrnují kombinování výstupů více AI workerů pro zlepšení celkové přesnosti nebo robustnosti systému. Místo spoléhání se na jediného AI workera využívají ensemblové techniky kolektivní inteligenci více workerů k činění informovanějších rozhodnutí.

Ensembly jsou obzvláště důležité v případech, kdy různé části vašeho pracovního postupu fungují nejlépe s různými AI modely, což je běžnější jev, než byste si mohli myslet. Výkonné modely jako GPT--4 jsou ve srovnání s méně schopnými open source alternativami extrémně drahé a pravděpodobně nejsou potřeba pro každý jednotlivý krok pracovního postupu vaší aplikace.

Běžnou ensemblovou technikou je většinové hlasování, kdy několik AI pracovníků nezávisle zpracovává stejný vstup a konečný výstup je určen většinovou shodou. Tento přístup může pomoci zmírnit dopad chyb jednotlivých pracovníků a zlepšit celkovou spolehlivost systému.

Představme si příklad, kde máme tři AI pracovníky pro analýzu sentimentu, přičemž každý používá jiný model nebo má k dispozici různé kontexty. Jejich výstupy můžeme kombinovat pomocí většinového hlasování pro určení konečné predikce sentimentu.

```
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
```

V tomto příkladu třída `SentimentAnalysisEnsemble` inicializuje text a vyvolává tři

různé AI pracovníky pro analýzu sentimentu. Metoda `analyze` shromažďuje predikce od každého pracovníka a určuje většinový sentiment pomocí metod `group_by` a `max_by`. Konečným výstupem je sentiment, který získá nejvíce hlasů od souboru pracovníků.

Soubory jsou jednoznačně případem, kdy může stát za to experimentovat s paralelismem.

Dynamický výběr a vyvolávání AI pracovníků

V některých, ne-li ve většině případů, může konkrétní AI pracovník, který má být vyvolán, záviset na běhových podmínkách nebo uživatelských vstupech. Dynamický výběr a vyvolávání AI pracovníků umožňují flexibilitu a adaptabilitu systému.

Možná budete v pokušení vtěsnat hodně funkcionality do jediného AI pracovníka a dát mu mnoho funkcí a velký komplikovaný prompt, který vysvětluje, jak je volat. Odolejte tomuto pokušení, věřte mi. Jedním z důvodů, proč se přístup, o kterém v této kapitole diskutujeme, nazývá "Množství pracovníků", je připomenout nám, že je žádoucí mít mnoho specializovaných pracovníků, z nichž každý dělá svou malou práci ve službě většímu účelu.

Například uvažujme chatbotovou aplikaci, kde různí AI pracovníci jsou zodpovědní za zpracování různých typů uživatelských dotazů. Na základě uživatelského vstupu aplikace dynamicky vybírá vhodného AI pracovníka pro zpracování dotazu.

```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
```

V tomto příkladu `ChatbotController` přijímá uživatelský dotaz prostřednictvím akce `process_query`. Nejprve použije `QueryClassifierWorker` k určení typu dotazu. Na základě klasifikovaného typu dotazu kontrolér dynamicky vybere vhodného AI pracovníka pro generování odpovědi. Tento dynamický výběr umožňuje chatbotovi zpracovávat různé typy dotazů a směrovat je k příslušným AI pracovníkům.

Vzhledem k tomu, že práce `QueryClassifierWorker` je relativně jednoduchá a nevyžaduje mnoho kontextu nebo definic funkcí, můžete ji pravděpodobně implementovat pomocí ultra-rychlého malého LLM jako `mistralai/mixtral-8x7b-instruct:nitro`. Jeho schopnosti se v mnoha úlohách blíží úrovni GPT--4 a v době, kdy toto píšu, ho Groq dokáže poskytovat s úžasnou propustností 444 tokenů za sekundu.

Kombinování tradičního NLP s LLM

Zatímco velké jazykové modely (LLM) způsobily revoluci v oblasti zpracování přirozeného jazyka (NLP) a nabízejí bezkonkurenční všestrannost a výkon v široké škále úloh, nejsou vždy nejefektivnějším nebo nákladově nejefektivnějším řešením každého problému. V mnoha případech může kombinace tradičních technik NLP s LLM vést k optimalizovanějším, cílenějším a ekonomičtějším přístupům k řešení specifických výzev NLP.

Představte si LLM jako švýcarské armádní nože NLP – neuvěřitelně všestranné a výkonné, ale ne nutně nejlepší nástroj pro každou práci. Někdy může být specializovaný nástroj jako vývrtka nebo otvírák na konzervy pro konkrétní úkol efektivnější. Podobně mohou tradiční techniky NLP, jako je shlukování dokumentů, identifikace témat a klasifikace, často poskytovat cílenější a nákladově efektivnější řešení pro určité aspekty vašeho NLP procesu.

Jednou z hlavních výhod tradičních technik NLP je jejich výpočetní efektivita. Tyto metody, které často spoléhají na jednodušší statistické modely nebo přístupy založené na pravidlech, mohou zpracovávat velké objemy textových dat mnohem rychleji a s nižší výpočetní náročností ve srovnání s LLM. To je činí obzvláště vhodnými pro úlohy zahrnující analýzu a organizaci velkých korpusů dokumentů, jako je shlukování podobných článků nebo identifikace klíčových témat v rámci kolekce textů.

Navíc tradiční techniky NLP mohou často dosáhnout vysoké přesnosti pro specifické úlohy, zejména když jsou trénovány na doménově specifických datasetech. Například dobře vyladěný klasifikátor dokumentů využívající tradiční algoritmy strojového učení jako Metoda podpůrných vektorů (SVM) nebo Naivní Bayes může přesně kategorizovat dokumenty do předem definovaných kategorií s minimálními výpočetními náklady.

LLM však skutečně vynikají v úlohách, které vyžadují hlubší porozumění jazyku, kontextu a uvažování. Jejich schopnost generovat koherentní a kontextově relevantní text, odpovídat na otázky a shrnovat dlouhé pasáže je nepřekonatelná tradičními

metodami NLP. LLM dokáží efektivně zpracovávat komplexní jazykové jevy, jako je nejednoznačnost, koreference a idiomatické výrazy, což je činí nepostradatelnými pro úlohy vyžadující generování přirozeného jazyka nebo porozumění.

Skutečná síla spočívá v kombinování tradičních technik NLP s LLM pro vytvoření hybridních přístupů, které využívají silné stránky obou. Použitím tradičních metod NLP pro úlohy jako předzpracování dokumentů, shlukování a extrakce témat můžete efektivně organizovat a strukturovat vaše textová data. Tyto strukturované informace pak mohou být předány LLM pro pokročilejší úlohy, jako je generování souhrnů, odpovídání na otázky nebo vytváření komplexních zpráv.

Například zvažme případ použití, kdy chcete vygenerovat zprávu o trendech pro specifickou doménu na základě velkého korpusu jednotlivých dokumentů o trendech. Místo spoléhání se pouze na LLM, což může být výpočetně náročné a časově náročné pro zpracování velkých objemů textu, můžete použít hybridní přístup:

1. Použijte tradiční techniky NLP, jako je modelování témat (např. Latentní Dirichletova alokace) nebo shlukovací algoritmy (např. K-means), pro seskupení podobných dokumentů o trendech a identifikaci klíčových témat v rámci korpusu.
2. Předejte shlukované dokumenty a identifikovaná témata do LLM, využívajíc jeho lepší porozumění jazyku a generativní schopnosti k vytvoření koherentních a informativních souhrnů pro každý shluk nebo téma.
3. Nakonec použijte LLM k vygenerování komplexní zprávy o trendech kombinováním jednotlivých souhrnů, zdůrazněním nejvýznamnějších trendů a poskytnutím vhledů a doporučení na základě agregovaných informací.

Kombinováním tradičních technik NLP s LLM tímto způsobem můžete efektivně zpracovávat velké množství textových dat, extrahovat smysluplné poznatky a generovat vysoce kvalitní zprávy při optimalizaci výpočetních zdrojů a nákladů.

Při zahájení vašich NLP projektů je zásadní pečlivě vyhodnotit specifické požadavky a omezení každého úkolu a zvážit, jak lze tradiční metody NLP a LLM společně využít

k dosažení nejlepších výsledků. Kombinací efektivity a přesnosti tradičních technik s všestranností a silou LLM můžete vytvářet vysoce účinná a ekonomická řešení NLP, která přinášejí hodnotu vašim uživatelům a zainteresovaným stranám.

Použití nástrojů

V oblasti vývoje aplikací založených na umělé inteligenci se koncept “použití nástrojů” nebo “volání funkcí” stal účinnou technikou, která umožňuje vašemu LLM připojit se k externím nástrojům, API, funkcím, databázím a dalším zdrojům. Tento přístup umožňuje bohatší škálu chování než pouhý výstup textu a dynamičtější interakce mezi vašimi AI komponenty a zbytkem ekosystému vaší aplikace. Jak se v této kapitole podíváme, použití nástrojů vám také dává možnost nechat váš AI model generovat data strukturovaným způsobem.

Co je použití nástrojů?

Použití nástrojů, také známé jako volání funkcí, je technika, která umožňuje vývojářům specifikovat seznam funkcí, se kterými může LLM během procesu generování pracovat. Tyto nástroje mohou sahat od jednoduchých pomocných funkcí až po komplexní API

nebo databázové dotazy. Poskytnutím přístupu k těmto nástrojům mohou vývojáři rozšířit schopnosti modelu a umožnit mu provádět úkoly, které vyžadují externí znalosti nebo akce.

obrázkem 8. Příklad definice funkce pro AI pracovníka, který analyzuje dokumenty

```
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
```

Klíčovou myšlenkou využití nástrojů je poskytnout LLM schopnost dynamicky vybírat a spouštět vhodné nástroje na základě vstupu uživatele nebo daného úkolu. Místo spoléhání se pouze na předtrénované znalosti modelu umožňuje využití nástrojů LLM využívat externí zdroje k generování přesnějších, relevantnějších a použitelnějších odpovědí. Využití nástrojů značně usnadňuje implementaci technik jako RAG

(Generování rozšířené o vyhledávání).

> Pokud není uvedeno jinak, tato kniha předpokládá, že váš AI model nemá přístup k žádným vestavěným nástrojům na straně serveru. Jakékoliv nástroje, které chcete zpřístupnit vašemu AI, musíte explicitně deklarovat v každém API požadavku, včetně ustanovení pro jejich spuštění, pokud a když vám AI sdělí, že by chtělo tento nástroj použít ve své odpovědi.

Potenciál využití nástrojů

Využití nástrojů otevírá širokou škálu možností pro aplikace řízené umělou inteligencí. Zde je několik příkladů toho, čeho lze dosáhnout pomocí využití nástrojů:

1. **Chatboti a virtuální asistenti**: Propojením LLM s externími nástroji mohou chatboti a virtuální asistenti provádět složitější úkoly, jako je získávání informací z databází, provádění API volání nebo interakce s jinými systémy. Například chatbot může pomocí nástroje CRM změnit stav obchodního případu na základě požadavku uživatele.
2. **Analýza dat a získávání poznatků**: LLM lze propojit s nástroji pro analýzu dat nebo knihovnami pro provádění pokročilých úloh zpracování dat. To umožňuje aplikacím generovat poznatky, provádět srovnávací analýzy nebo poskytovat doporučení založená na datech na základě uživatelských dotazů.
3. **Vyhledávání a získávání informací**: Využití nástrojů umožňuje LLM interagovat s vyhledávači, vektorovými databázemi nebo jinými systémy pro získávání informací. Transformací uživatelských dotazů na vyhledávací dotazy může LLM získávat relevantní informace z různých zdrojů a poskytovat komplexní odpovědi na uživatelské otázky.

4. **Integrace s externími službami**: Využití nástrojů umožňuje bezproblémovou integraci mezi aplikacemi řízenými AI a externími službami nebo API. Například LLM by mohlo komunikovat s API pro počasí, aby poskytovalo aktuální informace o počasí, nebo s API pro překlad, aby generovalo vícejazyčné odpovědi.

Pracovní postup při využití nástrojů

Pracovní postup při využití nástrojů typicky zahrnuje čtyři klíčové kroky:

1. Zahrnutí definic funkcí do kontextu požadavku
2. Dynamický (nebo explicitní) výběr nástrojů
3. Spuštění funkce/funkcí
4. Volitelné pokračování původního promptu

Pojďme si každý z těchto kroků podrobně projít.

Zahrnutí definic funkcí do kontextu požadavku

AI ví, jaké nástroje má k dispozici, protože jí poskytnete seznam jako součást vašeho požadavku na dokončení (typicky definovaný jako funkce pomocí varianty JSON schématu).

Přesná syntaxe definice nástroje je specifická pro každý model.

Takto definujete funkci `get_weather` v Claude 3:

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

A takto byste definovali stejnou funkci pro GPT--4, kdy ji předáte jako hodnotu parametru `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"],
```

```
    },
}
```

> Téměř stejné, až na to, že je to jiné bez zjevného důvodu! Jak otravné.

Definice funkcí určují název, popis a vstupní parametry. Vstupní parametry lze dále definovat pomocí atributů, jako jsou výčtové typy pro omezení přípustných hodnot, a určením, zda je parametr povinný či nikoliv.

Kromě samotných definic funkcí můžete do systémové direktivy zahrnout také pokyny nebo kontext vysvětlující, proč a jak funkci v systému používat.

Například můj nástroj Web Search v Olympii obsahuje tuto systémovou direktivu, která připomíná AI, že má zmíněné nástroje k dispozici:

```
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.
```

Poskytování podrobných popisů je považováno za nejdůležitější faktor ve výkonu nástroje. Vaše popisy by měly vysvětlovat každý detail o nástroji, včetně:

- Co nástroj dělá
- Kdy by měl být použit (a kdy ne)
- Co znamená každý parametr a jak ovlivňuje chování nástroje
- Veškeré důležité výhrady nebo omezení, která se vztahují k implementaci nástroje

Čím více kontextu poskytnete AI o vašich nástrojích, tím lépe bude schopna rozhodovat, kdy a jak je použít. Například Anthropic doporučuje pro svou řadu Claude 3 minimálně 3-4 věty na popis každého nástroje, více pokud je nástroj složitější.

Není to nutně intuitivní, ale popisy jsou považovány za důležitější než příklady. I když můžete do popisu nástroje nebo do doprovodného promptu zahrnout příklady jeho použití, je to méně důležité než mít jasné a komplexní vysvětlení účelu a parametrů nástroje. Příklady přidávejte až poté, co jste plně rozpracovali popis.

Zde je příklad specifikace API funkce podobné 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" ]
      }
    }
}
```

V praxi mají některé modely potíže se zpracováním vnořených specifikací funkcí a složitých výstupních datových typů, jako jsou pole, slovníky atd. Teoreticky byste však měli být schopni poskytovat specifikace JSON Schema libovolné hloubky!

Dynamický výběr nástrojů

Když spustíte chat completion, který obsahuje definice nástrojů, LLM dynamicky vybere nejvhodnější nástroj(e) k použití a vygeneruje požadované vstupní parametry pro každý nástroj.

V praxi je schopnost AI volat *přesně* správnou funkci a *přesně* dodržet vaši specifikaci vstupů různá. Snížení hyperparametru teploty až na 0.0 hodně pomáhá, ale podle mých zkušeností se stále občas objeví chyby. Tyto selhání zahrnují halucinované názvy funkcí, špatně pojmenované nebo zcela chybějící vstupní parametry. Parametry jsou předávány jako JSON, což znamená, že někdy uvidíte chyby způsobené zkrácením, špatným uvozením nebo jinak poškozeným JSONem.

Vzory Samoopravných dat mohou pomoci automaticky opravit volání funkcí, která se rozbijí kvůli syntaktickým chybám.

Vynucený (neboli explicitní) výběr nástrojů

Některé modely vám dávají možnost vynutit volání konkrétní funkce jako parametr v požadavku. V opačném případě je rozhodnutí o tom, zda funkci volat či nikoliv, zcela na uvážení AI.

Schopnost vynutit volání funkce je klíčová v určitých scénářích, kde chcete zajistit, aby byl spuštěn konkrétní nástroj nebo funkce, bez ohledu na proces dynamického výběru AI. Existuje několik důvodů, proč je tato schopnost důležitá:

1. **Explicitní kontrola**: Možná používáte AI jako *Diskrétní komponentu* nebo v předdefinovaném workflow, které vyžaduje provedení konkrétní funkce v konkrétním čase. Vynucením volání můžete zaručit, že požadovaná funkce bude vyvolána, místo toho, abyste museli AI zdvořile žádat o její provedení.
2. **Debugování a testování**: Při vývoji a testování aplikací řízených AI je schopnost vynutit volání funkcí neocenitelná pro účely debugování. Explicitním spouštěním specifických funkcí můžete izolovat a testovat jednotlivé komponenty vaší aplikace. To vám umožňuje ověřit správnost implementací funkcí, validovat vstupní parametry a zajistit, že jsou vráceny očekávané výsledky.
3. **Zvládání hraničních případů**: Mohou nastat hraniční případy nebo výjimečné scénáře, kdy proces dynamického výběru AI nemusí zvolit provedení funkce, kterou by měl, a vy to víte na základě externích procesů. V takových případech vám schopnost vynutit volání funkce umožňuje explicitně řešit tyto situace. Definujte pravidla nebo podmínky v logice vaší aplikace pro určení, kdy přepsat uvážení AI.
4. **Konzistence a reprodukovatelnost**: Pokud máte specifickou sekvenci funkcí, které je třeba provést v určitém pořadí, vynucení volání zaručuje, že stejná

sekvence bude dodržena pokaždé. To je zvláště důležité v aplikacích, kde jsou kritické konzistence a předvídatelné chování, například ve finančních systémech nebo vědeckých simulacích.

5. **Optimalizace výkonu**: V některých případech může vynucení volání funkce vést k optimalizaci výkonu. Pokud víte, že pro konkrétní úkol je vyžadována specifická funkce a že proces dynamického výběru AI by mohl způsobit zbytečnou režii, můžete obejít proces výběru a přímo vyvolat požadovanou funkci. To může pomoci snížit latenci a zlepšit celkovou efektivitu vaší aplikace.

Souhrnně řečeno, schopnost vynutit volání funkcí v aplikacích řízených AI poskytuje explicitní kontrolu, pomáhá při debugování a testování, zvládá hraniční případy a zajišťuje konzistenci a reprodukovatelnost. Je to mocný nástroj ve vašem arzenálu, ale musíme prodiskutovat ještě jeden aspekt této důležité funkce.

V mnoha případech rozhodování chceme, aby model vždy provedl volání funkce a nikdy neodpovídal pouze svými interními znalostmi. Například pokud směrujete mezi více modely specializovanými na různé úkoly (vícejazyčný vstup, matematika atd.), můžete použít model s voláním funkcí k delegování požadavků na jeden z pomocných modelů a nikdy neodpovídat samostatně.

Parametr výběru nástroje

GPT--4 a další jazykové modely, které podporují volání funkcí, vám poskytují parametr `tool_choice` pro řízení toho, zda je použití nástroje vyžadováno jako součást dokončení. Tento parametr má tři možné hodnoty:

- `auto` dává AI plnou volnost při používání nástroje nebo jednoduché odpovědi
- `required` říká AI, že *musí* zavolat nástroj *místo* odpovědi, ale ponechává výběr nástroje na AI

- Třetí možností je nastavit parametr `name_of_function`, který chcete vynutit. Více o tom v další části.

Všimněte si, že pokud nastavíte výběr nástroje (tool choice) na `required`, model bude nucen vybrat nejrelevantnější funkci k volání z těch, které mu byly poskytnuty, i když žádná z nich úplně neodpovídá zadání. V době publikace neznám žádný model, který by vrátil prázdnou odpověď `tool_calls` nebo jiným způsobem dal najevo, že nenašel vhodnou funkci k volání.

Vynucení funkce pro získání strukturovaného výstupu

Schopnost vynutit volání funkce vám dává způsob, jak získat strukturovaná data z chatovacího dokončování namísto toho, abyste je museli sami extrahovat z jeho textové odpovědi.

> Proč je vynucení funkcí pro získání strukturovaného výstupu tak důležité? Jednoduše proto, že extrakce strukturovaných dat z výstupu LLM je noční můra. Můžete si život trochu usnadnit tím, že požádáte o data v XML, ale pak musíte parsovat XML. A co uděláte, když to XML chybí, protože vaše AI odpověděla: "Omlouvám se, ale nemohu vygenerovat požadovaná data, protože bla, bla, bla..."

Při používání nástrojů tímto způsobem:

- Měli byste pravděpodobně definovat jediný nástroj ve vašem požadavku
- Nezapomeňte vynutit použití jeho funkce pomocí parametru `tool_choice`
- Pamatujte, že model předá vstup nástroji, takže název nástroje a popis by měly být z perspektivy modelu, ne vaší

Tento poslední bod si zaslouží vysvětlení na příkladu. Řekněme, že žádáte AI o analýzu sentimentu uživatelského textu. Název funkce by nebyl `analyze_sentiment`, ale spíše něco jako `save_sentiment_analysis`. AI je tou, která provádí analýzu sentimentu, *nikoliv nástroj*. Vše, co nástroj dělá (z pohledu AI), je ukládání výsledků analýzy.

Zde je příklad použití Claude 3 pro zaznamenání shrnutí obrázku do dobře strukturovaného JSON, tentokrát z příkazového řádku pomocí `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."
```

```
                    }
                ]
            }
        ]
}'
```

V uvedeném příkladu používáme model Claude 3 od společnosti Anthropic ke generování strukturovaného JSON souhrnu obrázku. Funguje to takto:

1. V datové části požadavku definujeme v poli `tools` jediný nástroj nazvaný `record_summary`. Tento nástroj je zodpovědný za zaznamenání souhrnu obrázku do dobře strukturovaného JSONu.
2. Nástroj `record_summary` má `input_schema`, které určuje očekávanou strukturu výstupu JSON. Definuje tři vlastnosti:
    - `key_colors`: Pole objektů představujících klíčové barvy v obrázku. Každý objekt barvy má vlastnosti pro hodnoty červené, zelené a modré (v rozsahu od 0.0 do 1.0) a člověkem čitelný název barvy ve formátu snake_case.
    - `description`: Vlastnost typu string pro stručný popis obrázku, omezený na 1-2 věty.
    - `estimated_year`: Volitelná vlastnost typu integer pro odhadovaný rok pořízení snímku, pokud se zdá být nefikční fotografií.
3. V poli `messages` poskytujeme obrazová data jako řetězec kódovaný ve formátu base64 spolu s typem média. To umožňuje modelu zpracovat obrázek jako součást vstupu.
4. Také dáváme Claudovi pokyn, aby použil nástroj `record_summary` k popisu obrázku.
5. Když je požadavek odeslán modelu Claude 3, analyzuje obrázek a vygeneruje souhrn JSON založený na specifikovaném `input_schema`. Model extrahuje klíčové barvy, poskytne stručný popis a odhadne rok pořízení snímku (pokud je to relevantní).

6. Vygenerovaný souhrn JSON je předán jako parametry nástroji `record_summary`, čímž poskytuje strukturovanou reprezentaci klíčových charakteristik obrázku.

Použitím nástroje `record_summary` s dobře definovaným `input_schema` můžeme získat strukturovaný JSON souhrn obrázku bez spoléhání na extrakci prostého textu. Tento přístup zajišťuje, že výstup sleduje konzistentní formát a může být snadno analyzován a zpracován následnými komponenty aplikace.

Schopnost vynutit volání funkce a specifikovat očekávanou strukturu výstupu je mocnou funkcí využití nástrojů v aplikacích řízených umělou inteligencí. Umožňuje vývojářům mít větší kontrolu nad generovaným výstupem a zjednodušuje integraci dat generovaných umělou inteligencí do pracovního postupu jejich aplikace.

Provedení funkce/funkcí

Definovali jste funkce a dali pokyn vaší AI, která se rozhodla, že by měla zavolat jednu z vašich funkcí. Nyní je čas, aby váš aplikační kód nebo knihovna, pokud používáte Ruby gem jako `raix-rails`, odeslaly volání funkce a její parametry do odpovídající implementace *ve vašem aplikačním kódu.*

Váš aplikační kód rozhodne, co dělat s výsledky provedení funkce. Možná to, co je třeba udělat, zahrnuje jediný řádek kódu v lambdě, nebo možná zahrnuje volání externího API. Možná to zahrnuje volání další AI komponenty, nebo možná zahrnuje stovky či dokonce tisíce řádků kódu ve zbytku vašeho systému. Je to zcela na vás.

Někdy je volání funkce koncem operace, ale pokud výsledky představují informace v řetězení myšlenek, které má AI dále zpracovávat, pak váš aplikační kód musí vložit výsledky provedení do přepisu chatu a nechat AI pokračovat ve zpracování.

Například zde je deklarace funkce Raix používaná Olympiiným `AccountManager` ke komunikaci s našimi klienty jako součást inteligentní orchestrace pracovních postupů pro zákaznický servis.

```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
```

Možná není na první pohled jasné, co se zde děje, tak si to rozebereme.

1. Třída `AccountManager` definuje mnoho funkcí souvisejících se správou účtů. Může měnit váš tarif, přidávat a odebírat členy týmu a mnoho dalšího.
2. Jeho instrukce na nejvyšší úrovni říkají `AccountManager`, že by měl informovat vlastníka účtu o výsledcích požadavku na změnu účtu pomocí funkce `notify_account_owner`.
3. Stručná definice funkce zahrnuje její:

- název
- popis
- parametry `message: { type: "string" }`
- blok kódu, který se má spustit při volání funkce

Po aktualizaci přepisu s výsledky funkčního bloku je znovu volána metoda `chat_completion`. Tato metoda je zodpovědná za odeslání aktualizovaného přepisu konverzace zpět do AI modelu pro další zpracování. Tento proces označujeme jako *konverzační smyčku*.

Když AI model obdrží nový požadavek na dokončení chatu s aktualizovaným přepisem, má přístup k výsledkům dříve provedené funkce. Může tyto výsledky analyzovat, začlenit je do svého rozhodovacího procesu a generovat další odpověď nebo akci na základě kumulativního kontextu konverzace. Na základě aktualizovaného kontextu může zvolit provedení dalších funkcí, nebo může vygenerovat konečnou odpověď na původní dotaz, pokud usoudí, že další volání funkcí není nutné.

Volitelné pokračování původního dotazu

Když pošlete výsledky nástroje zpět do LLM a pokračujete ve zpracování původního dotazu, AI použije tyto výsledky buď k volání dalších funkcí, nebo k vygenerování konečné textové odpovědi.

Některé modely, jako například Command-R od Cohere, mohou ve svých odpovědích citovat konkrétní nástroje, které použily, což poskytuje dodatečnou transparentnost a sledovatelnost.

V závislosti na použitém modelu budou výsledky volání funkce existovat v přepisových zprávách, které mají svou vlastní speciální roli, nebo se projeví v nějaké jiné syntaxi. Důležité však je, aby tato data byla v přepisu, aby je AI mohla zvážit při rozhodování o dalším postupu.

Častou (a potenciálně nákladnou) chybou je zapomenout přidat výsledky funkce do přepisu před pokračováním v chatu. V důsledku toho bude AI dotazována v podstatě stejným způsobem jako před prvním voláním funkce. Jinými slovy, z pohledu AI funkci ještě nevolala. Takže ji volá znovu. A znovu. A znovu, donekonečna, dokud ji nepřerušíte. Doufejme, že váš kontext nebyl příliš velký a váš model nebyl příliš drahý!

Osvědčené postupy pro používání nástrojů

Pro maximální využití nástrojů zvažte následující osvědčené postupy.

Popisné definice

Poskytněte jasné a popisné názvy a popisy pro každý nástroj a jeho vstupní parametry. To pomáhá LLM lépe porozumět účelu a možnostem každého nástroje.

> Z vlastní zkušenosti vám mohu říct, že běžná moudrost říkající, že "pojmenování je těžké", platí i zde; viděl jsem dramaticky odlišné výsledky od LLM jen změnou názvů funkcí nebo formulace popisů. Někdy odstranění popisů výkon dokonce *zlepší*.

Zpracování výsledků nástrojů

Při předávání výsledků nástrojů zpět do LLM zajistěte, aby byly dobře strukturované a komplexní. Používejte smysluplné klíče a hodnoty k reprezentaci výstupu každého nástroje. Experimentujte s různými formáty a zjistěte, který funguje nejlépe, od JSONu až po prostý text.

Interpretátor výsledků řeší tento problém využitím AI k analýze výsledků a poskytnutí vysvětlení, shrnutí nebo klíčových poznatků srozumitelných pro člověka.

Zpracování chyb

Implementujte robustní mechanismy pro zpracování chyb, které budou řešit případy, kdy LLM může generovat neplatné nebo nepodporované vstupní parametry pro volání

nástrojů. Elegantně zpracovávejte a zotavujte se z jakýchkoli chyb, které mohou během provádění nástroje nastat.

Jednou mimořádně příjemnou vlastností AI je, že rozumí chybovým hlášením! To znamená, že pokud pracujete v rychlém a méně precizním režimu, můžete jednoduše zachytit všechny výjimky generované při implementaci nástroje a předat je zpět AI, aby věděla, co se stalo!

Například zde je zjednodušená verze implementace vyhledávání Google v Olympii:

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

Vyhledávání Google v Olympii je dvoustupňový proces. Nejprve provedete vyhledávání a poté shrnete výsledky. Pokud dojde k jakékoliv chybě, je zpráva o výjimce zabalena a odeslána zpět umělé inteligenci. Tato technika je základem prakticky všech vzorů *Inteligentního zpracování chyb*.

Představme si například situaci, kdy volání API `GoogleSearch` selže kvůli výjimce 503 Service Unavailable. Ta se dostane až k nejvyšší úrovni zachycení chyb a popis chyby je odeslán zpět umělé inteligenci jako výsledek volání funkce. Místo toho, aby uživatel viděl prázdnou obrazovku nebo technickou chybu, umělá inteligence řekne něco jako "Omlouvám se, ale v tuto chvíli nemám přístup k vyhledávání Google. Mohu to zkusit později, pokud si přejete."

Může se to zdát jako pouhý chytrý trik, ale uvažujme o jiném druhu chyby, kdy umělá inteligence volá externí API a má přímou kontrolu nad parametry, které API předává.

Co když udělala chybu v tom, jak tyto parametry vygenerovala? Za předpokladu, že chybová zpráva z externího API je dostatečně podrobná, předání chybové zprávy zpět volající umělé inteligenci znamená, že může tyto parametry přehodnotit a zkusit to znovu. Automaticky. Bez ohledu na to, o jakou chybu šlo.

Nyní si představte, co by bylo potřeba k replikaci takového robustního zpracování chyb v *běžném* kódu. Je to prakticky nemožné.

Iterativní vylepšování

Pokud LLM nedoporučuje vhodné nástroje nebo generuje suboptimální odpovědi, iterujte definice nástrojů, popisů a vstupních parametrů. Průběžně vylepšujte a zdokonalujte nastavení nástrojů na základě pozorovaného chování a požadovaných výsledků.

1. Začněte s jednoduchými definicemi nástrojů: Začněte definováním nástrojů s jasnými a stručnými názvy, popisy a vstupními parametry. Zpočátku se vyvarujte příliš složitého nastavení nástrojů a soustřeďte se na základní funkcionalitu. Například pokud chcete ukládat výsledky analýzy sentimentu, začněte základní definicí jako:

```
{
  "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. Testujte a pozorujte: Jakmile máte počáteční definice nástrojů na místě, otestujte je s různými prompty a sledujte, jak LLM s nástrojem pracuje. Věnujte pozornost kvalitě a relevanci generovaných odpovědí. Pokud LLM generuje suboptimální odpovědi, je čas definice nástrojů vylepšit.
3. Upřesněte popis: Pokud LLM nechápe účel nástroje správně, zkuste upřesnit popis nástroje. Poskytněte více kontextu, příkladů nebo vysvětlení, která LLM navedou k efektivnímu používání nástroje. Například můžete aktualizovat popis nástroje pro analýzu sentimentu tak, aby konkrétněji adresoval *emoční zabarvení* analyzovaného textu:

```
{
  "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. Upravte vstupní parametry: Pokud LLM generuje neplatné nebo irelevantní vstupní parametry pro nástroj, zvažte úpravu definic parametrů. Přidejte specifičtější omezení, validační pravidla nebo příklady pro vyjasnění očekávaného vstupního formátu.

5. Iterujte na základě zpětné vazby: Průběžně sledujte výkon vašich nástrojů a shromažďujte zpětnou vazbu od uživatelů a zainteresovaných stran. Využijte tuto zpětnou vazbu k identifikaci oblastí pro zlepšení a provádějte iterativní vylepšení definic nástrojů. Například pokud uživatelé hlásí, že analýza správně nezpracovává sarkasmus, můžete do popisu přidat poznámku:

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

Iterativním vylepšováním definic vašich nástrojů na základě pozorovaného chování a zpětné vazby můžete postupně zlepšovat výkon a efektivitu vaší aplikace řízené umělou inteligencí. Nezapomeňte udržovat definice nástrojů jasné, stručné a zaměřené na konkrétní úkol. Pravidelně testujte a ověřujte interakce nástrojů, abyste zajistili, že odpovídají vašim požadovaným výsledkům.

Skládání a řetězení nástrojů

Jedním z nejsilnějších aspektů používání nástrojů, který byl dosud pouze naznačen, je schopnost skládat a řetězit více nástrojů dohromady pro plnění složitých úkolů. Pečlivým navržením definic vašich nástrojů a jejich vstupních/výstupních formátů můžete vytvořit znovupoužitelné stavební bloky, které lze kombinovat různými způsoby.

Uvažujme příklad, kde vytváříte pipeline pro analýzu dat pro vaši aplikaci řízenou umělou inteligencí. Můžete mít následující nástroje:

1. **`DataRetrieval`**: Nástroj, který získává data z databáze nebo API na základě stanovených kritérií.

2. **`DataProcessing`**: Nástroj, který provádí výpočty, transformace nebo agregace získaných dat.
3. **`DataVisualization`**: Nástroj, který prezentuje zpracovaná data v uživatelsky přívětivém formátu, jako jsou grafy nebo diagramy.

Řetězením těchto nástrojů můžete vytvořit výkonný workflow, který získává relevantní data, zpracovává je a prezentuje výsledky smysluplným způsobem. Takto by mohl vypadat workflow použití nástrojů:

1. LLM přijme uživatelský dotaz žádající o přehled prodejních dat pro specifickou kategorii produktů.
2. LLM vybere nástroj `DataRetrieval` a vygeneruje příslušné vstupní parametry pro získání relevantních prodejních dat z databáze.
3. Získaná data jsou "předána" nástroji `DataProcessing`, který vypočítá metriky jako celkový příjem, průměrnou prodejní cenu a míru růstu.
4. Zpracovaná data jsou pak zpracována nástrojem `DataVisualization`, který vytvoří vizuálně přitažlivý graf nebo diagram pro reprezentaci přehledu, předávající URL grafu zpět do LLM.
5. Nakonec LLM vygeneruje formátovanou odpověď na uživatelský dotaz pomocí markdownu, včetně vizualizovaných dat a shrnutí klíčových zjištění.

Skládáním těchto nástrojů dohromady můžete vytvořit plynulý workflow pro analýzu dat, který lze snadno integrovat do vaší aplikace. Krása tohoto přístupu spočívá v tom, že každý nástroj může být vyvíjen a testován nezávisle a pak kombinován různými způsoby k řešení různých problémů.

Pro umožnění plynulého skládání a řetězení nástrojů je důležité definovat jasné vstupní a výstupní formáty pro každý nástroj.

Například nástroj `DataRetrieval` může přijímat parametry jako jsou detaily připojení k databázi, název tabulky a podmínky dotazu a vracet výslednou sadu jako

strukturovaný JSON objekt. Nástroj `DataProcessing` pak může očekávat tento JSON objekt jako vstup a produkovat transformovaný JSON objekt jako výstup. Standardizací toku dat mezi nástroji můžete zajistit kompatibilitu a znovupoužitelnost.

Při navrhování vašeho ekosystému nástrojů přemýšlejte o tom, jak lze různé nástroje kombinovat pro řešení běžných případů použití ve vaší aplikaci. Zvažte vytvoření vysokoúrovňových nástrojů, které zapouzdřují běžné workflow nebo byznys logiku, což usnadní LLM jejich efektivní výběr a použití.

Pamatujte, že síla používání nástrojů spočívá ve flexibilitě a modularitě, kterou poskytuje. Rozdělením složitých úkolů na menší, znovupoužitelné nástroje můžete vytvořit robustní a adaptabilní aplikaci řízenou umělou inteligencí, která dokáže řešit širokou škálu výzev.

Budoucí směry

S vývojem oblasti vývoje aplikací řízených umělou inteligencí můžeme očekávat další pokroky v možnostech používání nástrojů. Některé potenciální budoucí směry zahrnují:

1. **Vícenásobné použití nástrojů**: LLM mohou být schopny rozhodnout, kolikrát potřebují použít nástroje k vygenerování uspokojivé odpovědi. To může zahrnovat více kol výběru a spouštění nástrojů na základě průběžných výsledků.
2. **Předdefinované nástroje**: AI platformy mohou poskytovat sadu předdefinovaných nástrojů, které mohou vývojáři využívat přímo, jako jsou Python interprety, nástroje pro vyhledávání na webu nebo běžné užitkové funkce.
3. **Bezproblémová integrace**: S rostoucí převahou používání nástrojů můžeme očekávat lepší integraci mezi AI platformami a populárními vývojovými frameworky, což vývojářům usnadní začlenění používání nástrojů do jejich aplikací.

Používání nástrojů je výkonná technika, která umožňuje vývojářům využít plný potenciál LLM v aplikacích řízených umělou inteligencí. Propojením LLM s externími nástroji a zdroji můžete vytvářet dynamičtější, inteligentnější a kontextově uvědomělé systémy, které se dokáží přizpůsobit potřebám uživatelů a poskytovat cenné přehledy a akce.

Zatímco používání nástrojů nabízí obrovské možnosti, je důležité být si vědom potenciálních výzev a úvah. Jedním klíčovým aspektem je správa složitosti interakcí nástrojů a zajištění stability a spolehlivosti celkového systému. Musíte řešit scénáře, kdy volání nástrojů může selhat, vrátit neočekávané výsledky nebo mít dopad na výkon. Kromě toho byste měli zvážit bezpečnostní opatření a kontrolu přístupu, abyste zabránili neoprávněnému nebo škodlivému použití nástrojů. Pro udržení integrity a výkonu vaší aplikace řízené umělou inteligencí jsou klíčové správné mechanismy pro zpracování chyb, logování a monitoring.

Při zkoumání možností použití nástrojů ve vašich vlastních projektech nezapomeňte začít s jasnými cíli, navrhněte dobře strukturované definice nástrojů a provádějte iterace na základě zpětné vazby a výsledků. Se správným přístupem a způsobem uvažování může použití nástrojů odemknout nové úrovně inovací a hodnoty ve vašich aplikacích řízených umělou inteligencí

Zpracování proudu dat

Streamování dat přes HTTP, také známé jako server-sent events (SSE), je mechanismus, kdy server průběžně odesílá data klientovi, jakmile jsou k dispozici, bez nutnosti, aby si je klient výslovně vyžádal. Vzhledem k tomu, že odpověď umělé inteligence je generována postupně, je logické poskytovat responzivní uživatelskou zkušenost zobrazováním výstupu UI v průběhu jeho generování. A ve skutečnosti všechna API poskytovatelů UI, které znám, nabízejí streamované odpovědi jako možnost ve svých dokončovacích endpointech.

Důvod, proč se tato kapitola objevuje v knize právě zde, hned po Používání nástrojů, je síla kombinace využití nástrojů s živými odpověďmi UI uživatelům. To umožňuje vytvářet dynamické a interaktivní zážitky, kde UI může zpracovávat uživatelské vstupy, využívat různé nástroje a funkce podle svého uvážení a poskytovat odpovědi v reálném čase.

Pro dosažení této plynulé interakce potřebujete napsat obsluhu proudu dat, která dokáže distribuovat volání nástrojových funkcí vyvolaných UI i běžný textový výstup koncovému uživateli. Potřeba cyklického zpracování po provedení nástrojové funkce přidává této úloze zajímavou výzvu.

Implementace ReplyStream

Pro demonstraci způsobu implementace zpracování proudu dat se tato kapitola podrobně zaměří na zjednodušenou verzi třídy `ReplyStream`, která se používá v systému Olympia. Instance této třídy lze předat jako parametr `stream` v knihovnách AI klientů, jako jsou ruby-openai a openrouter.

Zde je ukázka, jak používám `ReplyStream` v Olympiině `PromptSubscriber`, který pomocí Wisper naslouchá vytváření nových uživatelských zpráv.

```
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...
```

Kromě `context` odkazu na odběratele promptu, který ji vytvořil, obsahuje třída `ReplyStream` také instanční proměnné pro ukládání vyrovnávací paměti přijatých dat a pole pro sledování názvů funkcí a argumentů volaných během zpracování streamu.

```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` nastavuje počáteční stav instance `ReplyStream`, inicializuje vyrovnávací paměť, kontext a další proměnné.

Metoda `call` je hlavním vstupním bodem pro zpracování streamovaných dat. Přijímá `chunk` dat (reprezentovaný jako haš) a volitelný parametr `bytesize`, který v našem příkladu není využit. Uvnitř této metody třída používá porovnávání vzorů pro zpracování různých scénářů na základě struktury přijatého bloku dat.

Volání `deep_symbolize_keys` na bloku dat umožňuje elegantnější porovnávání vzorů tím, že můžeme pracovat se symboly místo řetězců.

```
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
```

První vzor, který porovnáváme, je volání nástroje spolu s jeho přidruženým názvem funkce. Pokud jej detekujeme, uložíme ho do pole `f_name`. Názvy funkcí ukládáme do indexovaného pole, protože model je schopen paralelního volání funkcí, kdy odesílá více funkcí k provedení najednou.

> Paralelní volání funkcí je schopnost AI modelu provádět více volání funkcí současně, což umožňuje zpracovat účinky a výsledky těchto volání funkcí paralelně. To je zvláště užitečné, pokud funkce trvají dlouhou dobu, a snižuje počet cest tam a zpět s rozhraním API, což může vést k významné úspoře spotřeby tokenů.

Dále potřebujeme najít shodu pro argumenty odpovídající voláním funkcí.

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

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

Podobně jako v případě názvů funkcí, ukládáme argumenty do indexovaného pole.

Dále sledujeme běžné zprávy zobrazované uživateli, které budou přicházet ze serveru po jednom tokenu a budou přiřazeny do proměnné `new_content`. Také musíme sledovat `finish_reason`. Ten bude mít hodnotu `nil` až do poslední části výstupní sekvence.

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

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

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

Je důležité, že přidáváme výraz pro porovnávání vzorů ke zpracování chybových zpráv zaslaných poskytovatelem AI modelu. V lokálních vývojových prostředích vyvoláme výjimku, ale v produkčním prostředí chybu zaznamenáme a dokončíme.

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

Závěrečná větev else příkazu case se provede, pokud žádný z předchozích vzorů neodpovídá. Je to jen pojistka pro případ, že by nám AI model začal posílat nerozpoznané části.

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

Metoda `send_to_client` je zodpovědná za odesílání obsahu uloženého ve vyrovnávací paměti klientovi. Kontroluje, zda není vyrovnávací paměť prázdná, aktualizuje obsah zprávy bota, vykresluje zprávu bota a ukládá obsah do databáze pro zajištění perzistence dat.

```
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` je volána po dokončení streamového zpracování. Zpracovává volání funkcí, pokud během streamu nějaká nastala, aktualizuje zprávu bota s konečným obsahem a dalšími relevantními informacemi a resetuje historii volání funkcí

```
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
```

Pokud se model rozhodne zavolat funkci, musíte toto volání funkce (název a argumenty) "zpracovat" takovým způsobem, aby se provedlo a zprávy `function_call` a `function_result` byly přidány do přepisu konverzace.

Podle mých zkušeností je lepší řešit vytváření funkčních zpráv na jednom místě ve vaší kódové základně, než spoléhat na implementace jednotlivých nástrojů. Je to nejen čistší řešení, ale má to i velmi důležitý praktický důvod: pokud model umělé inteligence zavolá funkci a při dalším průchodu neuvidí v přepisu výsledné volání a výsledek, *zavolá stejnou funkci znovu.* Potenciálně donekonečna. Nezapomeňte, že umělá inteligence je zcela bezstavová, takže pokud jí tyto volání funkcí nezpětně neukážete, jako by se nikdy nestaly.

```ruby
# PromptSubscriber#dispatch

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

Vymazání historie volání funkcí po jejich vykonání je stejně důležité jako zajištění, aby se volání a výsledky dostaly do vašeho přepisu, abyste nevolali stále dokola stejné funkce při každém průchodu smyčkou.

“Konverzační smyčka”

Ve třídě `PromptSubscriber` používáme metodu `prompt` z modulu `PromptDeclarations` k definování chování konverzační smyčky. Parametr `until` je nastaven na `-> { bot_message.complete? }`, což znamená, že smyčka bude pokračovat, dokud nebude `bot_message` označen jako dokončený.

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

Kdy je ale `bot_message` označena jako dokončená? Pokud jste zapomněli, podívejte se zpět na řádek 13 metody `finalize`.

Pojďme si projít celou logiku streamového zpracování.

1. `PromptSubscriber` obdrží novou zprávu od uživatele prostřednictvím metody `message_created`, která je vyvolána publikačně-odběratelským systémem Wisper pokaždé, když koncový uživatel vytvoří nový prompt.
2. Třídní metoda `prompt` deklarativně definuje chování logiky chat completion pro `PromptSubscriber`. AI model provede chat completion s obsahem uživatelské zprávy, novou instancí `ReplyStream` jako parametrem stream a specifikovanou podmínkou smyčky.
3. AI model zpracuje prompt a začne generovat odpověď. Během streamování odpovědi je pro každý fragment dat vyvolána metoda `call` instance `ReplyStream`.
4. Pokud se AI model rozhodne zavolat nástrojovou funkci, název funkce a argumenty jsou extrahovány z fragmentu a uloženy do polí `f_name` a `f_arguments`.
5. Pokud AI model generuje obsah zobrazovaný uživateli, je tento obsah uložen do vyrovnávací paměti a odeslán klientovi pomocí metody `send_to_client`.
6. Jakmile je streamové zpracování dokončeno, je volána metoda `finalize`. Pokud byly během streamu vyvolány nějaké nástrojové funkce, jsou odeslány pomocí metody `dispatch` třídy `PromptSubscriber`.
7. Metoda `dispatch` přidá zprávu `function_call` do přepisu konverzace, spustí odpovídající nástrojovou funkci a přidá zprávu `function_result` do přepisu s výsledkem volání funkce.
8. Po odeslání nástrojových funkcí je historie volání funkcí vymazána, aby se zabránilo duplicitním voláním funkcí v následujících smyčkách.
9. Pokud nebyly vyvolány žádné nástrojové funkce, metoda `finalize` aktualizuje `bot_message` s konečným obsahem, označí ji jako dokončenou a odešle aktualizovanou zprávu klientovi.
10. Je vyhodnocena podmínka smyčky `-> { bot_message.complete? }`. Pokud není `bot_message` označena jako dokončená, smyčka pokračuje a původní prompt je znovu odeslán s aktualizovaným přepisem konverzace.

11. Kroky 3-10 se opakují, dokud není `bot_message` označena jako dokončená, což znamená, že AI model dokončil generování své odpovědi a není třeba provádět další nástrojové funkce.

Implementací této konverzační smyčky umožníte AI modelu zapojit se do obousměrné interakce s aplikací, provádět nástrojové funkce podle potřeby a generovat odpovědi zobrazované uživateli, dokud konverzace nedosáhne přirozeného závěru.

Kombinace streamového zpracování a konverzační smyčky umožňuje dynamické a interaktivní zkušenosti poháněné umělou inteligencí, kde AI model může zpracovávat uživatelské vstupy, využívat různé nástroje a funkce a poskytovat odpovědi v reálném čase na základě vyvíjejícího se kontextu konverzace.

Automatické pokračování

Je důležité být si vědom omezení výstupu AI. Většina modelů má maximální počet tokenů, které mohou generovat v jedné odpovědi, což je určeno parametrem `max_tokens`. Pokud AI model během generování odpovědi dosáhne tohoto limitu, náhle se zastaví a oznámí, že výstup byl oříznut.

Ve streamované odpovědi z API AI platformy můžete tuto situaci detekovat prozkoumáním proměnné `finish_reason` ve fragmentu. Pokud je `finish_reason` nastavena na `"length"` (nebo jinou klíčovou hodnotu specifickou pro model), znamená to, že model během generování dosáhl svého maximálního limitu tokenů a výstup byl předčasně ukončen.

Jedním ze způsobů, jak elegantně zvládnout tento scénář a poskytnout plynulou uživatelskou zkušenost, je implementovat mechanismus automatického pokračování ve vaší logice streamového zpracování. Přidáním porovnávání vzorů pro důvody ukončení související s délkou můžete zvolit smyčku a automaticky pokračovat ve výstupu tam, kde skončil.

Zde je záměrně zjednodušený příklad toho, jak můžete upravit metodu `call` ve třídě `ReplyStream` pro podporu automatického pokračování:

```
LENGTH_STOPS = %w[length MAX_TOKENS]

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

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

      buffer << new_content.to_s

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

    # ...
  end
end

private

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

V této upravené verzi, když `finish_reason` indikuje zkrácený výstup, místo ukončení proudu přidáme do přepisu dvojici zpráv bez finalizace, přesuneme původní uživatelsky viditelnou zprávu na "konec" přepisu aktualizací jejího atributu `created_at`, a pak necháme smyčku pokračovat, aby AI mohlo pokračovat tam, kde skončilo.

Pamatujte, že koncový bod AI dokončování je bezstavový. "Zná" pouze to, co mu sdělíte prostřednictvím přepisu. V tomto případě způsob, jakým AI sdělujeme, že bylo přerušeno, je přidáním "neviditelných" (pro koncového uživatele) zpráv do přepisu. Nezapomeňte však, že toto je záměrně zjednodušený příklad. Skutečná implementace by musela provádět další správu přepisu, aby se zajistilo, že neplýtváme tokeny a/nebo nemáteme AI duplikovanými zprávami asistenta v přepisu.

Skutečná implementace automatického pokračování by měla mít také takzvanou "logiku přerušovače" , aby se zabránilo nekontrolovanému zacyklení. Důvodem je, že při určitých typech uživatelských výzev a nízkém nastavení `max_tokens` by AI mohlo nekonečně pokračovat v generování uživatelsky viditelného výstupu.

> Mějte na paměti, že každá smyčka vyžaduje samostatný požadavek a každý požadavek znovu spotřebuje celý váš přepis. Při rozhodování, zda implementovat automatické pokračování ve vaší aplikaci, byste měli rozhodně zvážit kompromisy mezi uživatelskou zkušeností a využitím API. Automatické pokračování může být obzvláště nebezpečně drahé, zejména při používání prémiových komerčních modelů.

Závěr

Zpracování proudu je klíčovým aspektem při vytváření aplikací poháněných umělou inteligencí , které kombinují použití nástrojů s živými odpověďmi AI. Efektivním zpracováním streamovaných dat z API platforem umělé inteligence můžete poskytnout plynulou a interaktivní uživatelskou zkušenost, zpracovávat velké odpovědi, optimalizovat využití zdrojů a elegantně zvládat chyby.

Poskytnutá třída `Conversation::ReplyStream` demonstruje, jak lze implementovat zpracování proudu v Ruby aplikaci pomocí porovnávání vzorů a architektury řízené

událostmi. Pochopením a využitím technik zpracování proudu můžete odemknout plný potenciál integrace AI ve vašich aplikacích a poskytovat výkonné a poutavé uživatelské zážitky.

Samoopravná data

Samoopravná data představují účinný přístup k zajištění integrity, konzistence a kvality dat v aplikacích využitím schopností velkých jazykových modelů (LLM). Tato kategorie vzorů se zaměřuje na myšlenku využití umělé inteligence k automatické detekci, diagnostice a opravě datových anomálií, nekonzistencí nebo chyb, čímž snižuje zátěž vývojářů a udržuje vysokou úroveň spolehlivosti dat.

V jádru vzorů samoopravných dat je uznání skutečnosti, že data jsou životně důležitou součástí každé aplikace a zajištění jejich přesnosti a integrity je klíčové pro správné fungování a uživatelskou zkušenost aplikace. Správa a údržba kvality dat však může být složitým a časově náročným úkolem, zejména když aplikace rostou co do velikosti a komplexity. Zde přichází ke slovu síla umělé inteligence.

Ve vzorech samoopravných dat jsou AI workeři využíváni k průběžnému monitorování a analýze dat vaší aplikace. Tyto modely mají schopnost chápat a interpretovat vzory,

vztahy a anomálie v datech. Využitím svých schopností zpracování a porozumění přirozenému jazyku mohou identifikovat potenciální problémy nebo nekonzistence v datech a podniknout příslušné kroky k jejich nápravě.

Proces samoopravných dat typicky zahrnuje několik klíčových kroků:

1. **Monitorování dat**: AI workeři neustále sledují datové toky aplikace, databáze nebo úložné systémy a hledají jakékoli známky anomálií, nekonzistencí nebo chyb. Případně můžete aktivovat AI komponentu v reakci na výjimku.
2. **Detekce anomálií**: Když je zjištěn problém, AI worker podrobně analyzuje data, aby identifikoval konkrétní povahu a rozsah problému. To může zahrnovat detekci chybějících hodnot, nekonzistentních formátů nebo dat, která porušují předem definovaná pravidla či omezení.
3. **Diagnostika a oprava**: Jakmile je problém identifikován, AI worker využije své znalosti a porozumění datové doméně k určení vhodného postupu. To může zahrnovat automatickou opravu dat, doplnění chybějících hodnot nebo označení problému pro lidský zásah, pokud je to nutné.
4. **Průběžné učení (volitelné, závisí na případu použití)**: Když váš AI worker narazí na různé datové problémy a vyřeší je, může vytvářet výstupy popisující, co se stalo a jak reagoval. Tato metadata lze využít v procesech učení, které vám (a případně i základnímu modelu prostřednictvím dolaďování) umožní být v průběhu času efektivnější při identifikaci a řešení datových anomálií.

Automatickou detekcí a opravou datových problémů můžete zajistit, že vaše aplikace pracuje s vysoce kvalitními, spolehlivými daty. To snižuje riziko chyb, nekonzistencí nebo datových bugů ovlivňujících funkčnost aplikace nebo uživatelskou zkušenost.

Jakmile máte AI workery, kteří se starají o monitorování a opravu dat, můžete se soustředit na další kritické aspekty aplikace. To šetří čas a zdroje, které by jinak byly vynaloženy na manuální čištění a údržbu dat. Ve skutečnosti, jak vaše aplikace rostou co do velikosti a komplexity, manuální správa kvality dat se stává stále náročnější. Vzory

“Samoopravných dat” efektivně škálují využitím síly AI ke zpracování velkých objemů dat a detekci problémů v reálném čase.

Díky své povaze se AI modely mohou adaptovat na měnící se datové vzory, schémata nebo požadavky v průběhu času s minimální nebo žádnou supervizí. Pokud jejich direktivy poskytují adekvátní vedení, zejména ohledně zamýšlených výsledků, vaše aplikace může být schopna se vyvíjet a zvládat nové datové scénáře bez nutnosti rozsáhlých manuálních zásahů nebo změn kódu.

Vzory samoopravných dat dobře ladí s ostatními kategoriemi vzorů, o kterých jsme diskutovali, jako je “Množství workerů”. Schopnost samoopravných dat lze vnímat jako specializovaný typ workera, který se zaměřuje specificky na zajištění kvality a integrity dat. Tento typ workera funguje společně s ostatními AI workery, přičemž každý přispívá k různým aspektům funkčnosti aplikace.

Implementace vzorů samoopravných dat v praxi vyžaduje pečlivý návrh a integraci AI modelů do architektury aplikace. Kvůli rizikům ztráty a poškození dat byste měli definovat jasné pokyny pro používání této techniky. Měli byste také zvážit faktory jako výkon, škálovatelnost a bezpečnost dat.

Praktická případová studie: Oprava poškozeného JSONu

Jeden z nejpraktičtějších a nejsnadněji vysvětlitelných způsobů využití samoopravných dat je také velmi jednoduchý: oprava poškozeného JSONu.

Tuto techniku lze aplikovat na běžnou výzvu řešení nedokonalých nebo nekonzistentních dat generovaných LLM, jako je poškozený JSON, a poskytuje přístup k automatické detekci a opravě těchto problémů.

V Olympii se pravidelně setkávám se situacemi, kdy LLM generují JSON data, která nejsou zcela validní. K tomu může docházet z různých důvodů, například když LLM přidá komentář před nebo za samotný JSON kód, nebo když zavede syntaktické chyby jako chybějící čárky či neescapované dvojité uvozovky. Tyto problémy mohou vést k chybám parsování a způsobit narušení funkčnosti aplikace.

Pro řešení tohoto problému jsem implementoval praktické řešení v podobě třídy JsonFixer. Tato třída ztělesňuje vzor "Samoopravných dat" tím, že přijímá poškozený JSON jako vstup a s využitím LLM ho opravuje, přičemž zachovává co nejvíce informací a původního záměru.

```
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
```

Všimněte si, jak `JsonFixer` používá Ventriloquist k usměrňování odpovědí AI.

Proces samoopravy JSON dat funguje následovně:

1. **Generování JSON**: K vytvoření JSON dat na základě určitých promptů nebo požadavků se používá LLM. Vzhledem k povaze LLM však generovaný JSON nemusí být vždy perfektně validní. JSON parser samozřejmě vyvolá `ParserError`, pokud mu předáte nevalidní JSON.

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

Všimněte si, že chybová zpráva je také předána volání `JSONFixer`, takže nemusí plně předpokládat, co je s daty špatně, zejména když parser často přesně řekne, kde je chyba.

2. **Oprava pomocí LLM**: Třída `JSONFixer` odešle poškozený JSON zpět do LLM, spolu se specifickým pokynem nebo instrukcí k opravě JSONu při maximálním zachování původních informací a záměru. LLM, trénovaný na obrovském množství dat a s porozuměním syntaxi JSONu, se pokusí opravit chyby a vygenerovat platný JSON řetězec. Pro omezení výstupu LLM se používá Response Fencing a jako AI model volíme Mixtral 8x7B, protože je pro tento typ úlohy obzvláště vhodný.
3. **Validace a integrace**: Opravený JSON řetězec vrácený LLM je parsován přímo třídou `JSONFixer`, protože byla volána funkce `chat_completion(json: true)`. Pokud opravený JSON projde validací, je integrován zpět do pracovního postupu aplikace, což umožňuje aplikaci plynule pokračovat ve zpracování dat. Poškozený JSON byl "vyléčen".

Přestože jsem napsal a přepsal svou vlastní implementaci `JSONFixer` několikrát, pochybuji, že celkový čas investovaný do všech těchto verzí přesahuje hodinu nebo dvě.

Všimněte si, že zachování záměru je klíčovým prvkem jakéhokoliv vzoru samoopravných dat. Proces opravy založený na LLM se snaží co nejvíce zachovat původní informace a záměr vygenerovaného JSONu. To zajišťuje, že opravený JSON si zachovává svůj sémantický význam a může být efektivně využit v kontextu aplikace.

Tato praktická implementace přístupu "samoopravných dat" v Olympii jasně ukazuje, jak lze využít AI, konkrétně LLM, k řešení skutečných datových výzev. Demonstruje sílu kombinace tradičních programovacích technik s možnostmi AI pro vytváření robustních a efektivních aplikací.

Postelův zákon a vzor "samoopravných dat"

"Samoopravná data", jak je představuje třída JSONFixer, dobře odpovídají principu známému jako Postelův zákon, také označovanému jako princip robustnosti. Postelův zákon říká:

"Buď konzervativní v tom, co děláš, a liberální v tom, co přijímáš od ostatních."

Tento princip, původně formulovaný Jonem Postelem, průkopníkem raného Internetu, zdůrazňuje důležitost budování systémů, které jsou tolerantní k různorodým nebo dokonce mírně nesprávným vstupům, zatímco při odesílání výstupů striktně dodržují stanovené protokoly.

V kontextu "samoopravných dat" třída JSONFixer ztělesňuje Postelův zákon tím, že je liberální v přijímání poškozeného nebo nedokonalého JSON dat generovaných LLM. Nezamítne okamžitě ani neselže při setkání s JSONem, který přísně neodpovídá očekávanému formátu. Místo toho zaujímá tolerantní přístup a pokouší se JSON opravit pomocí síly LLM.

Tím, že je liberální v přijímání nedokonalého JSONu, třída JSONFixer prokazuje robustnost a flexibilitu. Uznává, že data v reálném světě často přicházejí v různých formách a nemusí vždy odpovídat přísným specifikacím. Díky elegantnímu zvládání

a opravování těchto odchylek třída zajišťuje, že aplikace může plynule fungovat i v přítomnosti nedokonalých dat.

Na druhou stranu třída JSONFixer také dodržuje konzervativní aspekt Postelova zákona, pokud jde o výstup. Po opravě JSONu pomocí LLM třída validuje opravený JSON, aby zajistila, že přísně odpovídá očekávanému formátu. Zachovává integritu a správnost dat před jejich předáním dalším částem aplikace. Tento konzervativní přístup zaručuje, že výstup třídy JSONFixer je spolehlivý a konzistentní, podporuje interoperabilitu a brání šíření chyb.

Zajímavosti o Jonu Postelovi:

- Jon Postel (1943-1998) byl americký informatik, který hrál klíčovou roli ve vývoji Internetu. Byl znám jako "Bůh Internetu" pro své významné příspěvky k základním protokolům a standardům.
- Postel byl editorem série dokumentů Request for Comments (RFC), což je série technických a organizačních poznámek o Internetu. Je autorem nebo spoluautorem více než 200 RFC, včetně základních protokolů jako TCP, IP a SMTP.
- Kromě svých technických příspěvků byl Postel známý svým pokorným a kooperativním přístupem. Věřil v důležitost dosahování konsenzu a společné práce na budování robustní a interoperabilní sítě.
- Postel působil jako ředitel Divize počítačových sítí v Information Sciences Institute (ISI) na University of Southern California (USC) od roku 1977 až do své předčasné smrti v roce 1998.
- Za své ohromné příspěvky byl Postel posmrtně oceněn prestižní Turingovou cenou v roce 1998, často označovanou jako "Nobelova cena za informatiku."

Třída `JSONFixer` podporuje robustnost, flexibilitu a interoperabilitu, což byly základní hodnoty, kterých se Postel držel po celou svou kariéru. Budováním systémů, které jsou tolerantní k nedokonalostem při současném striktním dodržování

protokolů, můžeme vytvářet aplikace, které jsou odolnější a přizpůsobivější při řešení skutečných výzev.

Úvahy a kontraindikace

Použitelnost přístupů samoopravných dat zcela závisí na typu dat, se kterými vaše aplikace pracuje. Existuje důvod, proč možná nebudete chtít jednoduše upravit `JSON.parse` tak, aby automaticky opravoval *všechny chyby parsování JSONu* ve vaší aplikaci: ne všechny chyby lze nebo by měly být automaticky opraveny.

Samooprava je obzvláště problematická ve spojení s regulatorními požadavky nebo požadavky na shodu souvisejícími se zpracováním a manipulací s daty. Některá odvětví, jako je zdravotnictví a finance, mají tak přísné předpisy týkající se integrity dat a auditovatelnosti, že jakákoli "black box" oprava dat bez řádného dohledu nebo protokolování může tyto předpisy porušovat. Je zásadní zajistit, aby jakékoli techniky samoopravných dat, které vymyslíte, byly v souladu s příslušnými právními a regulačními rámci.

Aplikace technik samoopravných dat, zejména těch využívajících modely AI, může mít také významný dopad na výkon aplikace a využití zdrojů. Zpracování velkých objemů dat pomocí modelů AI pro detekci a opravu chyb může být výpočetně náročné. Je důležité vyhodnotit kompromisy mezi přínosy samoopravných dat a souvisejícími náklady na výkon a zdroje.

Pojďme se tedy ponořit do faktorů, které je třeba zvážit při rozhodování, kdy a kde tento mocný přístup použít.

Kritičnost dat

Při zvažování aplikace technik samoopravných dat je zásadní posoudit kritičnost zpracovávaných dat. Úroveň kritičnosti se vztahuje k důležitosti a citlivosti dat v kontextu vaší aplikace a její obchodní domény.

V některých případech nemusí být automatická oprava chyb v datech vhodná, zejména pokud jsou data vysoce citlivá nebo mají právní důsledky. Zvažte například následující scénáře:

1. **Finanční transakce**: Ve finančních aplikacích, jako jsou bankovní systémy nebo obchodní platformy, je přesnost dat nanejvýš důležitá. I drobné chyby ve finančních datech mohou mít významné důsledky, jako jsou nesprávné zůstatky na účtech, chybně směrované prostředky nebo chybná obchodní rozhodnutí. V těchto případech mohou automatické opravy bez důkladného ověření a auditu přinášet nepřijatelná rizika.
2. **Zdravotní záznamy**: Zdravotnické aplikace pracují s vysoce citlivými a důvěrnými údaji pacientů. Nepřesnosti ve zdravotních záznamech mohou mít vážné důsledky pro bezpečnost pacientů a rozhodnutí o léčbě. Automatická úprava zdravotních údajů bez řádného dohledu a validace kvalifikovanými zdravotnickými pracovníky může porušovat regulatorní požadavky a ohrozit pohodu pacientů.
3. **Právní dokumenty**: Aplikace zpracovávající právní dokumenty, jako jsou smlouvy, dohody nebo soudní podání, vyžadují přísnou přesnost a integritu. I drobné chyby v právních datech mohou mít významné právní důsledky. Automatické opravy v této oblasti nemusí být vhodné, protože data často vyžadují ruční kontrolu a ověření právními experty k zajištění jejich platnosti a vymahatelnosti.

V těchto kritických datových scénářích rizika spojená s automatickými opravami často převažují nad potenciálními přínosy. Důsledky zavedení chyb nebo nesprávné úpravy

dat mohou být závažné a vést k finančním ztrátám, právní odpovědnosti nebo dokonce poškození jednotlivců.

Při práci s vysoce kritickými daty je nezbytné upřednostnit procesy ručního ověřování a validace. Lidský dohled a odbornost jsou zásadní pro zajištění přesnosti a integrity dat. Automatizované techniky samoopravy lze stále využít k označení potenciálních chyb nebo nesrovnalostí, ale konečné rozhodnutí o opravách by mělo zahrnovat lidský úsudek a schválení.

Je však důležité poznamenat, že ne všechna data v aplikaci musí mít stejnou úroveň kritičnosti. V rámci stejné aplikace mohou existovat podmnožiny dat, které jsou méně citlivé nebo mají menší dopad, pokud dojde k chybám. V takových případech lze techniky samoopravných dat selektivně aplikovat na tyto specifické podmnožiny dat, zatímco kritická data zůstávají předmětem ručního ověřování.

Klíčové je pečlivě posoudit kritičnost každé kategorie dat ve vaší aplikaci a definovat jasné pokyny a procesy pro zpracování oprav na základě souvisejících rizik a důsledků. Rozlišováním mezi kritickými (tj. účetními knihami, zdravotními záznamy) a nekritickými daty (tj. poštovními adresami, varováními o zdrojích) můžete najít rovnováhu mezi využitím výhod technik samoopravných dat tam, kde je to vhodné, a udržením přísné kontroly a dohledu tam, kde je to nezbytné.

V konečném důsledku by rozhodnutí o aplikaci technik samoopravných dat na kritická data mělo být učiněno po konzultaci s oborovými experty, právními poradci a dalšími relevantními zainteresovanými stranami. Je nezbytné zvážit specifické požadavky, předpisy a rizika spojená s daty vaší aplikace a podle toho sladit strategie opravy dat.

Závažnost chyb

Při aplikaci technik samoopravných dat je důležité posoudit závažnost a dopad chyb v datech. Ne všechny chyby jsou si rovny a vhodný postup se může lišit v závislosti na závažnosti problému.

Drobné nesrovnalosti nebo problémy s formátováním mohou být vhodné pro automatickou opravu. Například pracovník pro samoopravná data pověřený opravou poškozeného JSONu může zpracovat chybějící čárky nebo neescapované dvojité uvozovky bez významného změnění významu nebo struktury dat. Tyto typy chyb lze často jednoduše opravit a mají minimální dopad na celkovou integritu dat.

Závažnější chyby, které zásadně mění význam nebo integritu dat, však mohou vyžadovat odlišný přístup. V takových případech nemusí být automatizované opravy dostačující a může být nutný lidský zásah, aby byla zajištěna přesnost a platnost dat.

Zde přichází na řadu koncept využití samotné umělé inteligence k určení závažnosti chyb. Využitím schopností modelů umělé inteligence můžeme navrhnout samoopravné datové pracovníky, kteří nejen opravují chyby, ale také vyhodnocují jejich závažnost a činí informovaná rozhodnutí o tom, jak s nimi naložit.

Představme si například samoopravného datového pracovníka zodpovědného za opravu nesrovnalostí v datech proudících do zákaznické databáze. Pracovníka lze navrhnout tak, aby analyzoval data a identifikoval potenciální chyby, jako jsou chybějící nebo protichůdné informace. Místo automatické opravy všech chyb však může být pracovník vybaven dodatečnými voláními nástrojů, které mu umožní označit závažné chyby k lidskému přezkoumání.

Zde je příklad, jak lze toto implementovat:

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

  attr_accessor :customer

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

  def initialize(customer)
    self.customer = customer
  end

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

        < additional instructions here... >

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

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

    self.stop = ["```"]

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

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

V tomto příkladu je worker `CustomerDataHealer` navržen k identifikaci a opravě nekonzistencí v zákaznických datech. Opět používáme Ohraničení odpovědí a Ventriloquist k získání strukturovaného výstupu. Je důležité, že systémová direktiva

workeru obsahuje instrukce k použití funkce `flag_for_review` v případě nalezení závažných chyb.

Když worker zpracovává zákaznická data, analyzuje je a pokouší se opravit případné nekonzistence. Pokud worker zjistí, že chyby jsou závažné a vyžadují lidský zásah, může použít nástroj `flag_for_review` k označení dat a poskytnutí důvodu pro toto označení.

Metoda `chat_completion` je volána s parametrem `json: true` pro parsování opravených zákaznických dat jako JSON. Není zde žádné ustanovení pro smyčku po volání funkce, takže výsledek bude prázdný, pokud byla vyvolána funkce `flag_for_review`. V opačném případě jsou data zákazníka aktualizována zkontrolovanými a potenciálně opravenými daty.

Začleněním hodnocení závažnosti chyb a možnosti označit data pro lidskou kontrolu se samoopravný datový worker stává inteligentnější a přizpůsobivější. Může automaticky zpracovávat menší chyby a zároveň eskalovat závažné chyby lidským expertům pro manuální zásah.

Konkrétní kritéria pro určení závažnosti chyb mohou být definována v direktivě workeru na základě znalosti domény a obchodních požadavků. Při posuzování závažnosti lze brát v úvahu faktory jako dopad na integritu dat, potenciál ztráty nebo poškození dat a důsledky nesprávných dat.

Využitím umělé inteligence k posouzení závažnosti chyb a poskytnutím možností pro lidský zásah mohou samoopravné datové techniky najít rovnováhu mezi automatizací a zachováním přesnosti dat. Tento přístup zajišťuje, že menší chyby jsou efektivně opraveny, zatímco závažné chyby získají potřebnou pozornost a odbornost od lidských kontrolorů.

Doménová komplexita

Při zvažování aplikace samoopravných datových technik je důležité vyhodnotit komplexitu datové domény a pravidla řídící její strukturu a vztahy. Komplexita

domény může významně ovlivnit efektivitu a proveditelnost automatizovaných přístupů k opravě dat.

Samoopravné datové techniky fungují dobře, když data sledují dobře definované vzory a omezení. V doménách, kde je datová struktura relativně jednoduchá a vztahy mezi datovými prvky jsou přímočaré, lze automatizované opravy aplikovat s vysokou mírou jistoty. Například oprava problémů s formátováním nebo vynucování základních omezení datových typů může být často efektivně řešena samoopravnými datovými workery.

Nicméně, jak se zvyšuje komplexita datové domény, rostou i výzvy spojené s automatizovanou opravou dat. V doménách se složitou obchodní logikou, komplexními vztahy mezi datovými entitami nebo doménově specifickými pravidly a výjimkami nemusí samoopravné datové techniky vždy zachytit všechny nuance a mohou zavést nezamýšlené důsledky.

Uvažme příklad komplexní domény: finanční obchodní systém. V této doméně data zahrnují různé finanční nástroje, tržní data, obchodní pravidla a regulační požadavky. Vztahy mezi různými datovými prvky mohou být složité a pravidla řídící platnost a konzistenci dat mohou být vysoce specifická pro danou doménu.

V takto komplexní doméně by samoopravný datový worker pověřený opravou nekonzistencí v obchodních datech musel mít hluboké pochopení doménově specifických pravidel a omezení. Musel by zvažovat faktory jako tržní regulace, obchodní limity, výpočty rizik a postupy vypořádání. Automatizované opravy v tomto kontextu nemusí vždy zachytit plnou komplexitu domény a mohou neúmyslně zavést chyby nebo porušit doménově specifická pravidla.

Pro řešení výzev doménové komplexity mohou být samoopravné datové techniky vylepšeny začleněním doménově specifických znalostí a pravidel do AI modelů a workerů. Toho lze dosáhnout pomocí technik jako:

1. **Doménově specifický trénink**: AI modely používané pro samoopravná data mohou být směrovány nebo dokonce doladěny na doménově specifických

datasetech, které zachycují složitosti a pravidla konkrétní domény. Vystavením modelů reprezentativním datům a scénářům se mohou naučit vzory, omezení a výjimky specifické pro doménu.

2. **Pravidly řízená omezení**: Samoopravní datoví workeři mohou být rozšířeni o explicitní pravidly řízená omezení, která kódují doménově specifické znalosti. Tato pravidla mohou být definována doménovými experty a integrována do procesu opravy dat. AI modely pak mohou tato pravidla využívat k vedení svých rozhodnutí a zajištění souladu s doménově specifickými požadavky.
3. **Spolupráce s doménovými experty**: V komplexních doménách je zásadní zapojit doménové experty do návrhu a vývoje samoopravných datových technik. Doménoví experti mohou poskytnout cenné vhledy do složitostí dat, obchodních pravidel a potenciálních hraničních případů. Jejich znalosti mohou být začleněny do AI modelů a workerů pro zlepšení přesnosti a spolehlivosti automatizovaných datových oprav pomocí vzorů Člověk v procesu.
4. **Inkrementální a iterativní přístup**: Při práci s komplexními doménami je často přínosné adoptovat inkrementální a iterativní přístup k samoopravným datům. Místo pokusu o automatizaci oprav pro celou doménu najednou se zaměřte na specifické subdomény nebo datové kategorie, kde jsou pravidla a omezení dobře pochopena. Postupně rozšiřujte rozsah samoopravných technik, jak roste porozumění doméně a techniky se prokazují jako efektivní.

Při zvážení složitosti datové domény a začlenění oborově specifických znalostí do technik samoopravných dat můžete dosáhnout rovnováhy mezi automatizací a přesností. Je důležité si uvědomit, že samoopravná data nejsou univerzálním řešením a že přístup by měl být přizpůsoben specifickým požadavkům a výzvám každé domény.

V komplexních doménách může být nejefektivnější hybridní přístup, který kombinuje techniky samoopravných dat s lidskou expertízou a dohledem. Automatické opravy mohou zpracovávat rutinní a dobře definované případy, zatímco komplexní scénáře nebo výjimky mohou být označeny pro lidskou kontrolu a zásah. Tento spolupracující

přístup zajišťuje, že výhody automatizace jsou realizovány při zachování nezbytné kontroly a přesnosti v komplexních datových doménách.

Vysvětlitelnost a transparentnost

Vysvětlitelnost se týká schopnosti porozumět a interpretovat důvody rozhodnutí učiněných modely umělé inteligence, zatímco transparentnost zahrnuje poskytování jasné viditelnosti do procesu opravy dat.

V mnoha kontextech musí být úpravy dat auditovatelné a odůvodnitelné. Zainteresované strany, včetně obchodních uživatelů, auditorů a regulačních orgánů, mohou vyžadovat vysvětlení, proč byly provedeny určité opravy dat a jak k těmto rozhodnutím modely umělé inteligence dospěly. To je zvláště důležité v oblastech, kde má přesnost a integrita dat významné důsledky, jako jsou finance, zdravotnictví a právní záležitosti.

Pro řešení potřeby vysvětlitelnosti a transparentnosti by měly techniky samoopravných dat zahrnovat mechanismy, které poskytují vhled do rozhodovacího procesu modelů umělé inteligence. Toho lze dosáhnout různými přístupy:

1. **Řetězec myšlení**: Požádání modelu, aby "nahlas" vysvětlil své uvažování před aplikací změn dat, může umožnit snazší pochopení rozhodovacího procesu a může generovat lidsky čitelná vysvětlení provedených oprav. Kompromisem je o něco větší složitost při oddělování vysvětlení od strukturovaného datového výstupu, což lze řešit…
2. **Generování vysvětlení**: Pracovníci se samoopravnými daty mohou být vybaveni schopností generovat lidsky čitelná vysvětlení oprav, které provádějí. Toho lze dosáhnout tím, že model bude požádán o výstup svého rozhodovacího procesu jako snadno srozumitelná vysvětlení *integrovaná přímo do dat.* Například pracovník se samoopravnými daty by mohl generovat zprávu, která zvýrazní konkrétní datové nesrovnalosti, které identifikoval, opravy, které aplikoval, a důvody těchto oprav.

3. **Důležitost vlastností**: Modely umělé inteligence mohou být instruovány informacemi o důležitosti různých vlastností nebo atributů v procesu opravy dat jako součást jejich směrnic. Tyto směrnice pak mohou být zpřístupněny lidským zainteresovaným stranám. Identifikací klíčových faktorů, které ovlivňují rozhodnutí modelu, mohou zainteresované strany získat vhled do důvodů oprav a posoudit jejich platnost.
4. **Protokolování a auditování**: Implementace komplexních mechanismů protokolování a auditování je klíčová pro zachování transparentnosti v procesu samoopravných dat. Každá oprava dat provedená modely umělé inteligence by měla být zaznamenána, včetně původních dat, opravených dat a konkrétních provedených akcí. Tato auditní stopa umožňuje retrospektivní analýzu a poskytuje jasný záznam o úpravách provedených v datech.
5. **Přístup s člověkem v procesu**: Začlenění přístupu s člověkem v procesu může zlepšit vysvětlitelnost a transparentnost technik samoopravných dat. Zapojením lidských expertů do kontroly a validace oprav generovaných umělou inteligencí mohou organizace zajistit, že opravy jsou v souladu s oborovými znalostmi a obchodními požadavky. Lidský dohled přidává další vrstvu odpovědnosti a umožňuje identifikaci potenciálních předpojatostí nebo chyb v modelech umělé inteligence.
6. **Kontinuální monitoring a hodnocení**: Pravidelné sledování a hodnocení výkonu technik samoopravných dat je nezbytné pro udržení transparentnosti a důvěry. Hodnocením přesnosti a efektivity modelů umělé inteligence v průběhu času mohou organizace identifikovat jakékoli odchylky nebo anomálie a přijmout nápravná opatření. Kontinuální monitoring pomáhá zajistit, že proces samoopravných dat zůstává spolehlivý a v souladu s požadovanými výsledky.

Vysvětlitelnost a transparentnost jsou kritickými aspekty při implementaci technik samoopravných dat. Poskytováním jasných vysvětlení pro opravy dat, udržováním komplexních auditních stop a zapojením lidského dohledu mohou organizace budovat

důvěru v proces samoopravných dat a zajistit, že úpravy provedené v datech jsou odůvodnitelné a v souladu s obchodními cíli.

Je důležité najít rovnováhu mezi výhodami automatizace a potřebou transparentnosti. Zatímco techniky samoopravných dat mohou významně zlepšit kvalitu dat a efektivitu, nemělo by to být na úkor ztráty viditelnosti a kontroly nad procesem opravy dat. Navrhováním pracovníků se samoopravnými daty s ohledem na vysvětlitelnost a transparentnost mohou organizace využít sílu umělé inteligence při zachování nezbytné úrovně odpovědnosti a důvěry v data.

Nezamýšlené důsledky

Zatímco techniky samoopravných dat mají za cíl zlepšit kvalitu a konzistenci dat, je zásadní být si vědom potenciálu nezamýšlených důsledků. Automatizované opravy, pokud nejsou pečlivě navrženy a monitorovány, mohou neúmyslně změnit význam nebo kontext dat, což vede k problémům v navazujících procesech.

Jedním z hlavních rizik samoopravných dat je zavádění předpojatosti nebo chyb v procesu opravy dat. Modely umělé inteligence, stejně jako jakýkoli jiný softwarový systém, mohou podléhat předpojatostem přítomným v trénovacích datech nebo zavedeným prostřednictvím návrhu algoritmů. Pokud tyto předpojatosti nejsou identifikovány a zmírněny, mohou se šířit procesem samoopravných dat a vést ke zkresleným nebo nesprávným úpravám dat.

Vezměme si například samoopravného datového pracovníka, jehož úkolem je opravovat nekonzistence v demografických údajích zákazníků. Pokud se AI model naučil předpojatosti z historických dat, jako je spojování určitých povolání nebo úrovní příjmů s konkrétním pohlavím či etnickou příslušností, může vytvářet nesprávné předpoklady a upravovat data způsobem, který tyto předsudky posiluje. To může vést k nepřesným profilům zákazníků, chybným obchodním rozhodnutím a potenciálně diskriminačním výsledkům.

Dalším možným nezamýšleným důsledkem je ztráta cenných informací nebo kontextu během procesu opravy dat. Techniky samoopravných dat se často zaměřují na standardizaci a normalizaci dat pro zajištění konzistence. V některých případech však mohou původní data obsahovat nuance, výjimky nebo kontextuální informace, které jsou důležité pro pochopení celkového obrazu. Automatizované opravy, které slepě vynucují standardizaci, mohou neúmyslně odstranit nebo zastřít tyto cenné informace.

Představte si například samoopravného datového pracovníka odpovědného za opravu nekonzistencí ve zdravotních záznamech. Pokud pracovník narazí na zdravotní anamnézu pacienta se vzácným onemocněním nebo neobvyklým léčebným plánem, může se pokusit normalizovat data tak, aby odpovídala běžnějšímu vzoru. Tím však může ztratit specifické detaily a kontext, které jsou klíčové pro přesné zachycení jedinečné situace pacienta. Tato ztráta informací může mít vážné důsledky pro péči o pacienta a lékařské rozhodování.

Pro zmírnění rizik nezamýšlených důsledků je nezbytné zaujmout proaktivní přístup při navrhování a implementaci technik samoopravných dat:

1. **Důkladné testování a validace**: Před nasazením samoopravných datových pracovníků do produkce je zásadní důkladně otestovat a ověřit jejich chování v různých scénářích. To zahrnuje testování s reprezentativními datovými sadami, které pokrývají různé hraniční případy, výjimky a potenciální předpojatosti. Důkladné testování pomáhá identifikovat a řešit případné nezamýšlené důsledky předtím, než ovlivní reálná data.
2. **Průběžné monitorování a hodnocení**: Implementace mechanismů průběžného monitorování a hodnocení je zásadní pro detekci a zmírnění nezamýšlených důsledků v průběhu času. Pravidelné přezkoumávání výsledků procesů samoopravných dat, analýza dopadu na navazující systémy a rozhodování a získávání zpětné vazby od zainteresovaných stran může pomoci identifikovat případné nežádoucí účinky a vyvolat včasná nápravná opatření. Pokud vaše organizace má provozní řídicí panely, je pravděpodobně dobrým nápadem přidat

jasně viditelné metriky související s automatizovanými změnami dat. Ještě lepším nápadem je pravděpodobně přidání alarmů spojených s velkými odchylkami od normální aktivity změn dat!

3. **Lidský dohled a intervence**: Udržování lidského dohledu a možnosti zasahovat do procesu samoopravných dat je klíčové. Přestože automatizace může výrazně zlepšit efektivitu, je důležité, aby lidští experti kontrolovali a validovali opravy provedené AI modely, zejména v kritických nebo citlivých oblastech. Lidský úsudek a odborné znalosti mohou pomoci identifikovat a řešit případné nezamýšlené důsledky, které mohou vzniknout.

4. **Vysvětlitelná AI (XAI) a transparentnost**: Jak bylo diskutováno v předchozí části, začlenění technik vysvětlitelné AI a zajištění transparentnosti v procesu samoopravných dat může pomoci zmírnit nezamýšlené důsledky. Poskytováním jasných vysvětlení pro opravy dat a udržováním komplexních auditních záznamů mohou organizace lépe porozumět a sledovat důvody úprav provedených AI modely.
5. **Inkrementální a iterativní přístup**: Přijetí inkrementálního a iterativního přístupu k samoopravným datům může pomoci minimalizovat riziko nezamýšlených důsledků. Místo aplikace automatizovaných oprav na celou datovou sadu najednou začněte s podmnožinou dat a postupně rozšiřujte rozsah, jak se techniky prokáží jako účinné a spolehlivé. To umožňuje pečlivé sledování a úpravy během procesu, čímž se snižuje dopad případných nezamýšlených důsledků.
6. **Spolupráce a zpětná vazba**: Zapojení zainteresovaných stran z různých oblastí a podpora spolupráce a zpětné vazby v průběhu procesu samoopravných dat může pomoci identifikovat a řešit nezamýšlené důsledky. Pravidelné získávání vstupů od odborníků v oboru, uživatelů dat a koncových uživatelů může poskytnout cenné poznatky o reálném dopadu oprav dat a upozornit na případné přehlédnuté problémy.

Proaktivním řešením rizika nezamýšlených důsledků a implementací vhodných bezpečnostních opatření mohou organizace využít výhod technik samoopravných dat při minimalizaci potenciálních nežádoucích účinků. Je důležité přistupovat k samoopravným datům jako k iterativnímu a kolaborativnímu procesu, neustále monitorovat, hodnotit a zdokonalovat techniky, aby byly v souladu s požadovanými výsledky a zachovávaly integritu a spolehlivost dat.

Při zvažování použití vzorů samoopravných dat je nezbytné pečlivě vyhodnotit tyto faktory a zvážit přínosy oproti potenciálním rizikům a omezením. V některých případech může být nejvhodnějším řešením hybridní přístup, který kombinuje automatizované opravy s lidským dohledem a intervencí.

Stojí také za zmínku, že techniky samoopravných dat by neměly být považovány za náhradu robustní validace dat, sanitizace vstupů a mechanismů zpracování chyb. Tyto základní postupy zůstávají kritické pro zajištění integrity a bezpečnosti dat. Samoopravná data by měla být vnímána jako doplňkový přístup, který může rozšířit a vylepšit tato existující opatření.

V konečném důsledku závisí rozhodnutí o použití vzorů samoopravných dat na konkrétních požadavcích, omezeních a prioritách vaší aplikace. Pečlivým zvážením výše uvedených aspektů a jejich sladěním s cíli a architekturou vaší aplikace můžete činit informovaná rozhodnutí o tom, kdy a jak efektivně využívat techniky samoopravných dat.

Kontextuální generování obsahu

Vzory kontextuálního generování obsahu využívají sílu velkých jazykových modelů (LLM) ke generování dynamického a kontextově specifického obsahu v aplikacích. Tato kategorie vzorů uznává důležitost poskytování personalizovaného a relevantního obsahu uživatelům na základě jejich konkrétních potřeb, preferencí a dokonce i předchozích a současných interakcí s aplikací.

> V kontextu tohoto přístupu se "obsahem" myslí jak primární obsah (tj. blogové příspěvky, články atd.), tak meta-obsah, jako jsou doporučení k primárnímu obsahu.

Vzory kontextuálního generování obsahu mohou hrát klíčovou roli při zvyšování úrovně zapojení uživatelů, poskytování přizpůsobených zážitků a automatizaci úkolů vytváření obsahu jak pro vás, tak pro vaše uživatele. Využitím vzorů, které popisujeme v této

kapitole, můžete vytvářet aplikace, které generují obsah dynamicky a přizpůsobují se kontextu a vstupům v reálném čase.

Vzory fungují integrací LLM do výstupů aplikace, od uživatelského rozhraní (někdy označovaného jako "chrome"), přes e-maily a další formy notifikací, až po jakékoli pipeline generování obsahu.

Když uživatel interaguje s aplikací nebo spustí konkrétní požadavek na obsah, aplikace zachytí relevantní kontext, jako jsou uživatelské preference, předchozí interakce nebo konkrétní podněty. Tyto kontextuální informace jsou pak spolu s případnými šablonami nebo pokyny předány do LLM a použity k vytvoření textového výstupu, který by jinak musel být buď napevno zakódován, uložen v databázi nebo algoritmicky generován.

Obsah generovaný pomocí LLM může mít různé formy, jako jsou personalizovaná doporučení, dynamické popisy produktů, přizpůsobené e-mailové odpovědi nebo dokonce celé články či blogové příspěvky. Jedním z nejradikálnějších využití tohoto obsahu, které jsem před více než rokem zavedl, je dynamické generování prvků uživatelského rozhraní, jako jsou popisky formulářů, nápovědy a další druhy vysvětlujícího textu.

Personalizace

Jednou z klíčových výhod vzorů kontextuálního generování obsahu je schopnost poskytovat vysoce personalizované zážitky uživatelům. Generováním obsahu založeného na kontextu specifickém pro uživatele tyto vzory umožňují aplikacím přizpůsobit obsah individuálním zájmům, preferencím a interakcím uživatelů.

Personalizace jde nad rámec pouhého vložení jména uživatele do obecného obsahu. Zahrnuje využití bohatého kontextu dostupného o každém uživateli k generování obsahu, který rezonuje s jejich specifickými potřebami a přáními. Tento kontext může zahrnovat širokou škálu faktorů, jako jsou:

1. **Informace z uživatelského profilu**: Na nejobecnější úrovni aplikace této techniky lze demografická data, zájmy, preference a další atributy profilu použít ke generování obsahu, který odpovídá uživatelovu zázemí a charakteristikám.
2. **Behaviorální data**: Předchozí interakce uživatele s aplikací, jako jsou zobrazené stránky, kliknuté odkazy nebo zakoupené produkty, mohou poskytnout cenné informace o jejich chování a zájmech. Tato data lze použít ke generování návrhů obsahu, který odráží jejich vzorce zapojení a předvídá jejich budoucí potřeby.
3. **Kontextové faktory**: Současný kontext uživatele, jako je jeho poloha, zařízení, denní doba nebo dokonce počasí, může ovlivnit proces generování obsahu. Například cestovní aplikace může mít AI pracovníka, který je schopen generovat personalizovaná doporučení na základě aktuální polohy uživatele a převládajících povětrnostních podmínek.

Využitím těchto kontextových faktorů umožňují vzory kontextuálního generování obsahu aplikacím poskytovat obsah, který působí jako šitý na míru každému jednotlivému uživateli. Tato úroveň personalizace má několik významných výhod:

1. **Zvýšené zapojení**: Personalizovaný obsah upoutává pozornost uživatelů a udržuje je zapojené do aplikace. Když uživatelé cítí, že obsah je relevantní a mluví přímo k jejich potřebám, je pravděpodobnější, že stráví více času interakcí s aplikací a prozkoumáváním jejích funkcí.
2. **Zlepšená spokojenost uživatelů**: Personalizovaný obsah ukazuje, že aplikace rozumí a záleží jí na jedinečných požadavcích uživatele. Poskytováním obsahu, který je užitečný, informativní a v souladu s jejich zájmy, může aplikace zvýšit spokojenost uživatelů a vybudovat s nimi silnější spojení.
3. **Vyšší míra konverze**: V kontextu e-commerce nebo marketingových aplikací může personalizovaný obsah významně ovlivnit míru konverze. Prezentováním produktů, nabídek nebo doporučení, které jsou přizpůsobeny preferencím a chování uživatelů, může aplikace zvýšit pravděpodobnost, že uživatelé provedou požadované akce, jako je nákup nebo registrace ke službě.

Produktivita

Vzory kontextuálního generování obsahu mohou výrazně zvýšit určité druhy produktivity tím, že snižují potřebu manuálního generování obsahu a úprav v kreativních procesech. Využitím síly LLM můžete generovat kvalitní obsah ve velkém měřítku a šetřit tak čas a úsilí, které by vaši tvůrci obsahu a vývojáři jinak museli věnovat zdlouhavé manuální práci.

Tradičně musí tvůrci obsahu zkoumat, psát, upravovat a formátovat obsah tak, aby splňoval požadavky aplikace a očekávání uživatelů. Tento proces může být časově náročný a vyžadovat značné zdroje, zejména s rostoucím objemem obsahu.

Nicméně s využitím vzorů kontextuálního generování obsahu lze proces tvorby obsahu z velké části automatizovat. Velké jazykové modely dokážou generovat souvislý, gramaticky správný a kontextově relevantní obsah na základě poskytnutých pokynů a vodítek. Tato automatizace přináší několik výhod pro produktivitu:

1. **Snížení manuální práce**: Díky delegování úkolů generování obsahu na velké jazykové modely se mohou tvůrci obsahu soustředit na úkoly vyšší úrovně, jako je obsahová strategie, tvorba nápadů a zajištění kvality. Mohou poskytnout modelu nezbytný kontext, šablony a pokyny a nechat ho zpracovat samotné generování obsahu. To snižuje množství manuální práce potřebné pro psaní a úpravy, což umožňuje tvůrcům obsahu být produktivnější a efektivnější.
2. **Rychlejší tvorba obsahu**: Velké jazykové modely dokážou generovat obsah mnohem rychleji než lidští autoři. Se správnými pokyny a vodítky může model vytvořit několik kusů obsahu během několika sekund či minut. Tato rychlost umožňuje aplikacím generovat obsah mnohem rychlejším tempem a držet krok s požadavky uživatelů a neustále se měnícím digitálním prostředím.

> Nevede rychlejší tvorba obsahu k situaci "tragédie obecní pastviny", kdy je internet zahlcen obsahem, který nikdo nečte? Bohužel se obávám, že odpověď je ano.

3. **Konzistence a kvalita**: Velké jazykové modely mohou snadno upravovat obsah tak, aby byl konzistentní ve stylu, tónu a kvalitě. S jasnými pokyny a příklady mohou určité typy aplikací (např. zpravodajské redakce, PR oddělení atd.) zajistit, že jejich obsah vytvořený člověkem odpovídá jejich firemnímu hlasu a splňuje požadované standardy kvality. Tato konzistence snižuje potřebu rozsáhlých úprav a revizí, čímž šetří čas a úsilí v procesu tvorby obsahu.
4. **Iterace a optimalizace**: Vzory kontextuálního generování obsahu umožňují rychlou iteraci a optimalizaci obsahu. Úpravou pokynů, šablon nebo vodítek poskytnutých modelu mohou vaše aplikace rychle generovat varianty obsahu a automatizovaně testovat různé přístupy způsobem, který v minulosti nebyl možný. Tento iterativní proces umožňuje rychlejší experimentování a vylepšování obsahových strategií, což vede k efektivnějšímu a poutavějšímu obsahu v průběhu času. Tato konkrétní technika může být zásadním průlomem pro aplikace jako je e-commerce, které stojí a padají na míře okamžitého opuštění a uživatelské angažovanosti.

Je důležité poznamenat, že přestože vzory kontextuálního generování obsahu mohou výrazně zvýšit produktivitu, zcela neodstraňují potřebu lidského zapojení. Tvůrci obsahu a editoři stále hrají klíčovou roli při definování celkové obsahové strategie, poskytování vedení modelu a zajišťování kvality a vhodnosti generovaného obsahu.

Automatizací více repetitivních a časově náročných aspektů tvorby obsahu vzory kontextuálního generování obsahu uvolňují cenný lidský čas a zdroje, které lze

přesměrovat na úkoly s vyšší hodnotou. Tato zvýšená produktivita vám umožňuje poskytovat uživatelům personalizovanější a poutavější obsah při současné optimalizaci pracovních postupů tvorby obsahu.

Rychlá iterace a experimentování

Vzory kontextuálního generování obsahu vám umožňují rychle iterovat a experimentovat s různými variantami obsahu, což umožňuje rychlejší optimalizaci a vylepšování vaší obsahové strategie. Můžete generovat více verzí obsahu během několika sekund, jednoduše úpravou kontextu, šablon nebo pokynů poskytnutých modelu.

Tato schopnost rychlé iterace přináší několik klíčových výhod:

1. **Testování a optimalizace**: Díky schopnosti rychle generovat varianty obsahu můžete snadno testovat různé přístupy a měřit jejich efektivitu. Například můžete generovat více verzí popisu produktu nebo marketingového sdělení, každou přizpůsobenou specifickému segmentu uživatelů nebo kontextu. Analyzováním metrik uživatelské angažovanosti, jako je míra prokliku nebo míra konverze, můžete identifikovat nejefektivnější varianty obsahu a podle toho optimalizovat vaši obsahovou strategii.

2. **A/B testování**: Vzory kontextuálního generování obsahu umožňují bezproblémové A/B testování obsahu. Můžete generovat dvě nebo více variant obsahu a náhodně je zobrazovat různým skupinám uživatelů. Porovnáním výkonu každé varianty můžete určit, který obsah nejlépe rezonuje s vaší cílovou skupinou. Tento přístup založený na datech vám umožňuje činit informovaná rozhodnutí a neustále vylepšovat váš obsah pro maximalizaci uživatelské angažovanosti a dosažení požadovaných výsledků.

3. **Personalizační experimenty**: Rychlá iterace a experimentování jsou obzvláště cenné, když přijde na personalizaci. Se vzory kontextuálního generování obsahu můžete rychle generovat personalizované varianty obsahu založené na různých uživatelských segmentech, preferencích nebo chování. Experimentováním s různými personalizačními strategiemi můžete identifikovat nejefektivnější přístupy pro zapojení jednotlivých uživatelů a poskytování přizpůsobených zážitků.
4. **Přizpůsobení se měnícím se trendům**: Schopnost rychlé iterace a experimentování vám umožňuje zůstat agilní a přizpůsobovat se měnícím se trendům a preferencím uživatelů. Když se objeví nová témata, klíčová slova nebo uživatelské chování, můžete rychle vytvořit obsah, který je s těmito trendy v souladu. Neustálým experimentováním a vylepšováním svého obsahu můžete zůstat relevantní a udržet si konkurenční výhodu v neustále se vyvíjející digitální krajině.
5. **Nákladově efektivní experimentování**: Tradiční experimentování s obsahem často vyžaduje značný čas a zdroje, protože tvůrci obsahu musí ručně vyvíjet a testovat různé varianty. S využitím vzorů Kontextového generování obsahu se však náklady na experimentování výrazně snižují. Velké jazykové modely dokáží rychle generovat varianty obsahu ve velkém měřítku, což vám umožňuje prozkoumat širokou škálu nápadů a přístupů bez významných nákladů.

Pro maximální využití rychlé iterace a experimentování je důležité mít zavedený dobře definovaný experimentální rámec. Tento rámec by měl zahrnovat:

- Jasné cíle a hypotézy pro každý experiment
- Vhodné metriky a sledovací mechanismy pro měření výkonnosti obsahu
- Strategie segmentace a cílení pro zajištění, že relevantní varianty obsahu jsou doručovány správným uživatelům
- Analytické a reportovací nástroje pro získávání poznatků z experimentálních dat
- Proces pro začlenění poznatků a optimalizací do vaší obsahové strategie

Přijetím rychlé iterace a experimentování můžete neustále vylepšovat a optimalizovat svůj obsah, zajišťovat, že zůstává poutavý, relevantní a efektivní při dosahování cílů vaší aplikace. Tento agilní přístup k tvorbě obsahu vám umožňuje být o krok napřed a poskytovat výjimečné uživatelské zážitky.

Škálovatelnost a efektivita

S růstem aplikací a rostoucí poptávkou po personalizovaném obsahu umožňují vzory kontextového generování obsahu efektivní škálování tvorby obsahu. Velké jazykové modely dokáží současně generovat obsah pro velký počet uživatelů a kontextů bez nutnosti proporcionálního navýšení lidských zdrojů. Tato škálovatelnost umožňuje aplikacím poskytovat personalizované zážitky rostoucí uživatelské základně bez přetížení jejich schopností tvorby obsahu.

Všimněte si, že kontextové generování obsahu lze efektivně využít k internacionalizaci vaší aplikace "za běhu". Ve skutečnosti je to přesně to, co jsem udělal pomocí mého Instant18n Gemu pro poskytování Olympie ve více než půl tuctu jazyků, i když jsme mladší než rok.

AI poháněná lokalizace

Pokud mi dovolíte se na chvíli pochlubit, myslím, že moje knihovna Instant18n pro Rails aplikace je průlomovým příkladem vzoru "Kontextového generování obsahu" v akci, který ukazuje transformační potenciál AI ve vývoji aplikací. Tento gem využívá sílu velkého jazykového modelu GPT od OpenAI k revoluci ve způsobu, jakým se řeší internacionalizace a lokalizace v Rails aplikacích.

Tradičně internacionalizace Rails aplikace zahrnuje ruční definování překladových klíčů a poskytování odpovídajících překladů pro každý podporovaný jazyk. Tento proces

může být časově náročný, náročný na zdroje a náchylný k nekonzistencím. S gemem Instant18n je však paradigma lokalizace zcela předefinováno.

Integrací velkého jazykového modelu umožňuje gem Instant18n generovat překlady za běhu, založené na kontextu a významu textu. Místo spoléhání se na předdefinované překladové klíče a statické překlady gem dynamicky překládá text pomocí síly AI. Tento přístup nabízí několik klíčových výhod:

1. **Bezproblémová lokalizace**: S gemem Instant18n již vývojáři nemusí ručně definovat a udržovat překladové soubory pro každý podporovaný jazyk. Gem automaticky generuje překlady na základě poskytnutého textu a požadovaného cílového jazyka, což činí proces lokalizace bezproblémovým a plynulým.
2. **Kontextová přesnost**: AI může dostat dostatek kontextu k pochopení nuancí překládaného textu. Může brát v úvahu okolní kontext, idiomy a kulturní odkazy k generování překladů, které jsou přesné, přirozeně znějící a kontextově vhodné.
3. **Rozsáhlá jazyková podpora**: Gem Instant18n využívá rozsáhlé znalosti a jazykové schopnosti GPT, umožňující překlady do široké škály jazyků. Od běžných jazyků jako španělština a francouzština až po obscurnější nebo fiktivní jazyky jako klingonština a elfština, gem zvládne širokou škálu překladových požadavků.
4. **Flexibilita a kreativita**: Gem překračuje hranice tradičních jazykových překladů a umožňuje kreativní a nekonvenční možnosti lokalizace. Vývojáři mohou překládat text do různých stylů, dialektů nebo dokonce fiktivních jazyků, což otevírá nové možnosti pro jedinečné uživatelské zážitky a poutavý obsah.
5. **Optimalizace výkonu**: Gem Instant18n obsahuje mechanismy ukládání do mezipaměti pro zlepšení výkonu a snížení režie opakovaných překladů. Přeložený text je ukládán do mezipaměti, což umožňuje rychlé obsloužení následných požadavků na stejný překlad bez nutnosti redundantních API volání.

Gem Instant18n představuje sílu vzoru "Kontextového generování obsahu" využitím AI k dynamickému generování lokalizovaného obsahu. Ukazuje, jak lze AI integrovat do

základní funkcionality Rails aplikace a transformovat způsob, jakým vývojáři přistupují k internacionalizaci a lokalizaci.

Díky odstranění potřeby manuální správy překladů a umožnění překladů za běhu na základě kontextu šetří gem Instant18n vývojářům významné množství času a úsilí. Umožňuje jim soustředit se na budování hlavních funkcí jejich aplikace a současně zajišťuje, že je lokalizace řešena plynule a přesně.

Význam uživatelského testování a zpětné vazby

Na závěr mějte vždy na paměti důležitost uživatelského testování a zpětné vazby. Je zásadní ověřit, že kontextové generování obsahu splňuje očekávání uživatelů a je v souladu s cíli aplikace. Průběžně iterujte a vylepšujte generovaný obsah na základě uživatelských postřehů a analytiky. Pokud generujete dynamický obsah ve velkém měřítku, který by bylo nemožné manuálně ověřit vámi a vaším týmem, zvažte přidání mechanismů zpětné vazby, které uživatelům umožní nahlásit obsah, který je zvláštní nebo nesprávný, spolu s vysvětlením proč. Tato cenná zpětná vazba může být dokonce předána AI pracovníkovi, který je pověřen úpravami komponenty, která obsah vygenerovala!

Generativní uživatelské rozhraní

Pozornost je v dnešní době natolik cenná, že efektivní zapojení uživatelů nyní vyžaduje softwarové zkušenosti, které jsou nejen bezproblémové a intuitivní, ale také vysoce personalizované podle individuálních potřeb, preferencí a kontextů. V důsledku toho designéři a vývojáři čelí stále častěji výzvě vytvářet uživatelská rozhraní, která se dokáží přizpůsobit a vyhovět jedinečným požadavkům každého uživatele *ve velkém měřítku*.

Generativní uživatelské rozhraní (GenUI) představuje skutečně revoluční přístup k návrhu uživatelského rozhraní, který využívá sílu velkých jazykových modelů (LLMs) k vytváření vysoce personalizovaných a dynamických uživatelských zážitků v reálném čase. Chtěl jsem se ujistit, že vám v této knize poskytnu alespoň základní informace o GenUI, protože věřím, že jde o jednu z nejzelenějších příležitostí, která v současnosti existuje v oblasti návrhu aplikací a frameworků. Jsem přesvědčen, že v tomto konkrétním odvětví vzniknou desítky či více úspěšných komerčních a open-source projektů.

V jádru GenUI kombinuje principy Kontextové generace obsahu s pokročilými technikami umělé inteligence k dynamickému generování prvků uživatelského rozhraní, jako jsou text, obrázky a rozložení, na základě hlubokého pochopení kontextu, preferencí a cílů uživatele. GenUI umožňuje designérům a vývojářům vytvářet rozhraní, která se přizpůsobují a vyvíjejí v reakci na interakce uživatelů, poskytující úroveň personalizace, která byla dříve nedosažitelná.

GenUI představuje zásadní změnu v přístupu k návrhu uživatelského rozhraní. Místo navrhování pro masy nám GenUI umožňuje navrhovat pro jednotlivce. Personalizovaný obsah a rozhraní mají potenciál vytvářet uživatelské zážitky, které rezonují s každým uživatelem na hlubší úrovni, zvyšují zapojení, spokojenost a loajalitu.

Jako špičková technologie je přechod na GenUI plný koncepčních a praktických výzev. Integrace umělé inteligence do procesu návrhu, zajištění, aby generovaná rozhraní byla nejen personalizovaná, ale také použitelná, přístupná a v souladu s celkovou značkou a uživatelskou zkušeností - to vše jsou výzvy, které činí GenUI záležitostí pro menšinu, nikoli většinu. Navíc zapojení umělé inteligence vyvolává otázky ohledně ochrany soukromí, transparentnosti a dokonce i etických důsledků.

Navzdory výzvám mají personalizované zážitky ve velkém měřítku sílu zcela transformovat způsob, jakým interagujeme s digitálními produkty a službami. Otevírá možnosti pro vytváření inkluzivních a přístupných rozhraní, která vyhovují různorodým potřebám uživatelů bez ohledu na jejich schopnosti, zázemí či preference.

V této kapitole prozkoumáme koncept GenUI, přičemž se zaměříme na některé definující charakteristiky, klíčové výhody a potenciální výzvy. Začneme nejzákladnější a nejdostupnější formou GenUI: generováním textového obsahu pro jinak tradičně navržená a implementovaná uživatelská rozhraní.

Generování textů pro uživatelská rozhraní

Textové prvky, které existují v rozhraní vaší aplikace, jako jsou popisky formulářů, nápovědy a vysvětlující texty, jsou typicky napevno zakódované do šablon nebo komponent UI, což poskytuje konzistentní, ale obecnou zkušenost pro všechny uživatele. Pomocí vzorů kontextové generace obsahu můžete transformovat tyto statické prvky na dynamické, kontextově uvědomělé a personalizované komponenty.

Personalizované formuláře

Formuláře jsou všudypřítomnou součástí webových a mobilních aplikací a slouží jako hlavní prostředek pro sběr uživatelských vstupů. Tradiční formuláře však často představují obecnou a neosobní zkušenost se standardními popisky a poli, které nemusí vždy odpovídat specifickému kontextu nebo potřebám uživatele. Uživatelé s větší pravděpodobností vyplní formuláře, které se jim zdají přizpůsobené jejich potřebám a preferencím, což vede k vyšší míře konverze a spokojenosti uživatelů.

Je však důležité najít rovnováhu mezi personalizací a konzistencí. Zatímco přizpůsobení formulářů jednotlivým uživatelům může být přínosné, je zásadní zachovat určitou míru známosti a předvídatelnosti. Uživatelé by měli být stále schopni snadno rozpoznat formuláře a orientovat se v nich, i když obsahují personalizované prvky.

Zde je několik nápadů na personalizované formuláře pro inspiraci:

Kontextové návrhy polí

GenUI může analyzovat předchozí interakce uživatele, preference a data k poskytování inteligentních návrhů polí jako předpovědí. Například pokud uživatel již dříve zadal svou doručovací adresu, formulář může automaticky vyplnit příslušná pole jejich uloženými informacemi. To nejen šetří čas, ale také ukazuje, že aplikace chápe a pamatuje si preference uživatele.

Počkat moment, není tahle technika něco, co by se dalo udělat i bez zapojení AI? Samozřejmě že ano, ale krása řízení takové funkcionality pomocí AI spočívá ve dvou věcech: 1) jak snadná může být implementace a 2) jak odolná může být vůči změnám a vývoji vašeho UI v průběhu času.

Pojďme si rychle sestavit službu pro náš teoretický systém zpracování objednávek, která se bude snažit proaktivně vyplnit správnou doručovací adresu uživatele.

```
class OrderShippingAddressSubscriber
  include Raix::ChatCompletion

  attr_accessor :order

  delegate :customer, to: :order

  DIRECTIVE = "You are a smart order processing assistant. Given the
  customer's order history, guess the most likely shipping address
  for the current order."

  def order_created(order)
    return unless order.pending? && order.shipping_address.blank?

    self.order = order

    transcript.clear
    transcript << { system: DIRECTIVE }
    transcript << { user: "Order History: #{order_history.to_json}" }
    transcript << { user: "Current Order: #{order.to_json}" }

    response = chat_completion
    apply_predicted_shipping_address(order, response)
  end

  private

  def apply_predicted_shipping_address(order, response)
    # extract the shipping address from the response...
    # ...and assume there's some sort of live update of the address fields
    order.update(shipping_address:)
  end
```

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

Tento příklad je velmi zjednodušený, ale měl by fungovat ve většině případů. Základní myšlenkou je nechat AI hádat stejným způsobem jako člověk. Abych lépe vysvětlil, o čem mluvím, podívejme se na nějaká vzorová data:

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

```
  {"date": "2024-05-02", "description": "infant toys",
   "shipping_address": "789 Suburb St, Quietville"},
  {"date": "2024-05-14", "description": "outdoor grill",
   "shipping_address": "123 Country Lane, Rural Town"},
  {"date": "2024-05-29", "description": "literary classics",
  "shipping_address": "456 City Apt, Metroville"},
  {"date": "2024-06-11", "description": "baby clothes",
   "shipping_address": "789 Suburb St, Quietville"},
  {"date": "2024-07-01", "description": "watering can",
   "shipping_address": "123 Country Lane, Rural Town"},
  {"date": "2024-07-18", "description": "non-fiction essays",
  "shipping_address": "456 City Apt, Metroville"},
  {"date": "2024-07-28", "description": "baby bath items",
  "shipping_address": "789 Suburb St, Quietville"},
  {"date": "2024-08-09", "description": "herb garden kit",
  "shipping_address": "123 Country Lane, Rural Town"},
  {"date": "2024-08-24", "description": "children's books",
  "shipping_address": "456 City Apt, Metroville"}
]
```

Všimli jste si toho vzoru v datech? Garantuji vám, že pro LLM je to hračka. Pojďme si to ukázat a zeptáme se GPT--4, jaká je nejpravděpodobnější doručovací adresa pro "teploměr".

```
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.
```

Pokud si říkáte, že je přehnané používat na tento úkol drahý model jako GPT--4, máte pravdu! Vyzkoušel jsem stejný prompt na Mistral 7B Instruct a ten vyprodukoval následující odpověď rychlostí 75 tokenů za sekundu, a to za skromnou cenu 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.
```

Stojí režie a náklady této techniky za to, aby byl zážitek z nakupování magičtější? Pro mnoho internetových prodejců rozhodně ano. A podle všeho budou náklady na AI výpočty jen klesat, zejména u poskytovatelů hostingu open-source modelů v závodu o nejnižší ceny.

Použijte Šablonu promptu a StructuredIO spolu s Ohraničením odpovědí k optimalizaci tohoto typu chatové komunikace.

Adaptivní řazení polí

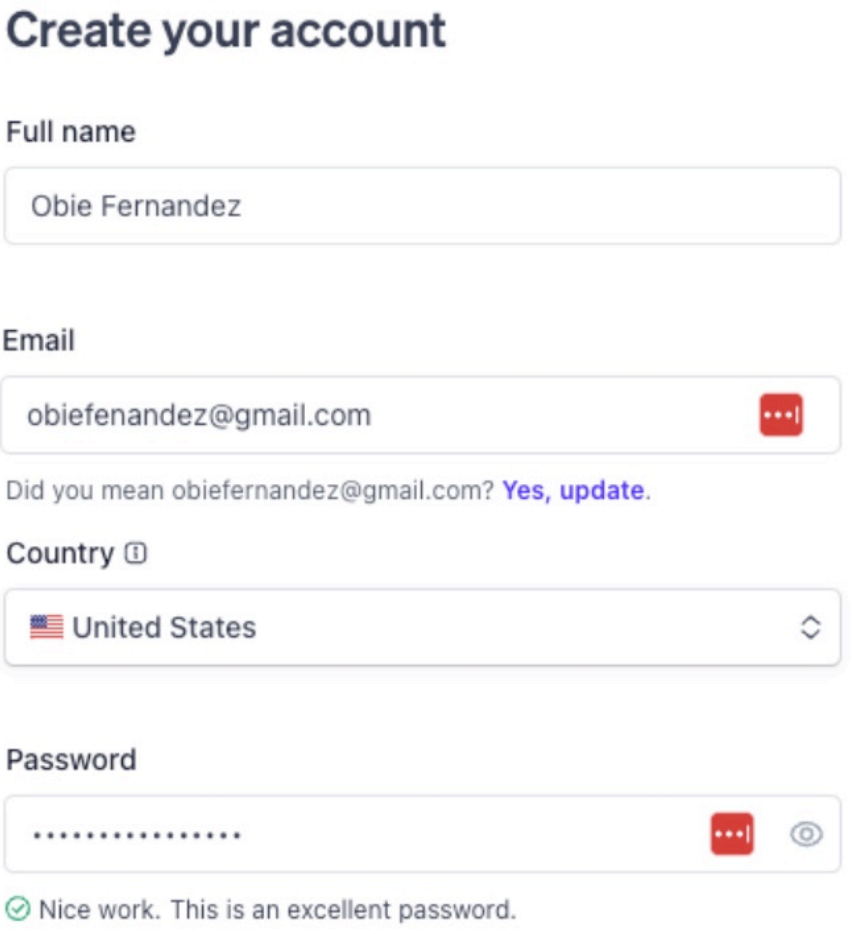
Pořadí, ve kterém jsou formulářová pole prezentována, může výrazně ovlivnit uživatelský zážitek a míru dokončení. S GenUI můžete dynamicky upravovat pořadí polí na základě kontextu uživatele a důležitosti každého pole. Například pokud uživatel vyplňuje registrační formulář pro fitness aplikaci, formulář může upřednostnit pole související s jejich fitness cíli a preferencemi, což činí proces relevantnější a poutavější.

Personalizované mikrotexty

Instrukční text, chybové zprávy a další mikrotexty spojené s formuláři lze také personalizovat pomocí GenUI. Místo zobrazování obecných chybových zpráv jako "Neplatná e-mailová adresa" můžete generovat užitečnější a kontextuální zprávy jako "Prosím zadejte platnou e-mailovou adresu pro přijetí potvrzení vaší objednávky." Tyto personalizované prvky mohou učinit práci s formulářem uživatelsky přívětivější a méně frustrující.

Personalizovaná validace

Podobně jako u Personalizovaných mikrotextů byste mohli použít AI k validaci formuláře způsobem, který působí magicky. Představte si, že necháte AI validovat formulář uživatelského profilu a hledat potenciální chyby na *sémantické* úrovni.



obrázkem 9. Dokážete rozpoznat probíhající sémantickou validaci?

Postupné odkrývání

GenUI může inteligentně určit, která formulářová pole jsou nezbytná na základě kontextu uživatele a postupně odkrývat další pole podle potřeby. Tato technika postupného odkrývání pomáhá snížit kognitivní zátěž a činí proces vyplňování formuláře zvládnutelnějším. Například pokud se uživatel přihlašuje k základnímu

předplatnému, formulář může nejprve zobrazit pouze nezbytná pole a jak uživatel postupuje nebo vybírá konkrétní možnosti, mohou být dynamicky představena další relevantní pole.

Kontextově citlivý vysvětlující text

Popisky se často používají k poskytnutí dodatečných informací nebo vedení uživatelů, když se pohybují nad konkrétními prvky nebo s nimi interagují. S přístupem "Generování kontextového obsahu" můžete vytvářet popisky, které se přizpůsobují kontextu uživatele a poskytují relevantní informace. Například když uživatel zkoumá složitou funkci, popisek může nabídnout personalizované tipy nebo příklady založené na jejich předchozích interakcích nebo úrovni dovedností.

Vysvětlující text, jako jsou instrukce, popisy nebo pomocné zprávy, může být dynamicky generován na základě kontextu uživatele. Místo prezentace obecných vysvětlení můžete použít LLM k generování textu, který je přizpůsoben specifickým potřebám nebo otázkám uživatele. Například pokud má uživatel potíže s konkrétním krokem v procesu, vysvětlující text může poskytnout personalizované vedení nebo tipy pro řešení problémů.

Mikrotexty označují malé kousky textu, které provázejí uživatele vaší aplikací, jako jsou popisky tlačítek, chybové zprávy nebo potvrzovací výzvy. Aplikováním přístupu Generování kontextového obsahu na mikrotexty můžete vytvořit adaptivní UI, které reaguje na akce uživatele a poskytuje relevantní a užitečný text. Například když se uživatel chystá provést kritickou akci, potvrzovací výzva může být dynamicky generována tak, aby poskytla jasnou a personalizovanou zprávu.

Personalizovaný vysvětlující text a popisky mohou výrazně vylepšit proces onboardingu nových uživatelů. Poskytováním kontextově specifického vedení a příkladů můžete uživatelům pomoci rychle porozumět aplikaci a navigovat v ní, což snižuje křivku učení a zvyšuje adopci.

Dynamické a kontextově citlivé prvky rozhraní mohou také způsobit, že aplikace působí intuitivněji a poutavěji. Uživatelé jsou více nakloněni interakci a zkoumání funkcí, když je doprovodný text přizpůsoben jejich specifickým potřebám a zájmům.

Dosud jsme se zabývali nápady na vylepšení stávajících paradigmat uživatelského rozhraní pomocí AI, ale co kdybychom radikálněji přehodnotili způsob, jakým jsou uživatelská rozhraní navrhována a implementována?

Definice generativního UI

Na rozdíl od tradičního návrhu UI, kde designéři vytvářejí pevná, statická rozhraní, GenUI naznačuje budoucnost, ve které náš software nabídne flexibilní, personalizované zážitky, které se mohou vyvíjet a přizpůsobovat v reálném čase. Pokaždé, když používáme konverzační rozhraní řízené AI, umožňujeme AI přizpůsobit se konkrétním potřebám uživatele. GenUI posouvá věci o krok dále tím, že aplikuje tuto úroveň přizpůsobivosti na *vizuální* rozhraní softwaru.

Důvodem, proč je dnes možné experimentovat s myšlenkami GenUI, je to, že velké jazykové modely již rozumí programování a jejich základní znalosti zahrnují technologie a frameworky UI. Otázkou tedy je, zda lze velké jazykové modely využít ke generování prvků UI, jako jsou text, obrázky, layouty a dokonce celá rozhraní, která jsou přizpůsobena každému jednotlivému uživateli. Model by mohl být instruován, aby bral v úvahu různé faktory, jako jsou předchozí interakce uživatele, uvedené preference, demografické informace a aktuální kontext použití, k vytvoření vysoce personalizovaných a relevantních rozhraní.

GenUI se od tradičního návrhu uživatelského rozhraní liší v několika klíčových aspektech:

1. **Dynamické a adaptivní**: Tradiční návrh UI zahrnuje vytváření pevných, statických rozhraní, která zůstávají stejná pro všechny uživatele. Naproti tomu GenUI umožňuje rozhraní, která se mohou dynamicky přizpůsobovat a měnit na základě potřeb uživatele a kontextu. To znamená, že stejná aplikace může prezentovat různá rozhraní různým uživatelům nebo dokonce stejnému uživateli v různých situacích.
2. **Personalizace ve velkém měřítku**: U tradičního designu je vytváření personalizovaných zážitků pro každého uživatele často nepraktické kvůli času a zdrojům, které by to vyžadovalo. GenUI naopak umožňuje personalizaci ve velkém měřítku. Využitím AI mohou designéři vytvářet rozhraní, která se automaticky přizpůsobují jedinečným potřebám a preferencím každého uživatele, aniž by museli ručně navrhovat a vyvíjet samostatná rozhraní pro každý segment uživatelů.
3. **Zaměření na výsledky**: Tradiční návrh UI se často zaměřuje na vytváření vizuálně přitažlivých a funkčních rozhraní. Zatímco tyto aspekty jsou důležité i v GenUI, primární zaměření se přesouvá k dosahování požadovaných uživatelských výsledků. GenUI se snaží vytvářet rozhraní, která jsou optimalizována pro specifické cíle a úkoly každého uživatele, přičemž upřednostňuje použitelnost a efektivitu před čistě estetickými úvahami.
4. **Kontinuální učení a zlepšování**: Systémy GenUI se mohou průběžně učit a zlepšovat na základě interakcí uživatelů a zpětné vazby. Když uživatelé pracují s generovanými rozhraními, AI modely mohou shromažďovat data o chování uživatelů, preferencích a výsledcích a využívat tyto informace k vylepšování a optimalizaci budoucích generací rozhraní. Tento iterativní proces učení umožňuje systémům GenUI stávat se postupem času stále efektivnějšími v plnění potřeb uživatelů.

Je důležité poznamenat, že GenUI není totéž co nástroje pro design s podporou AI, jako jsou ty, které poskytují návrhy nebo automatizují určité designové úkoly. Zatímco tyto nástroje mohou být užitečné při zefektivnění procesu návrhu, stále spoléhají na

designéry, kteří činí konečná rozhodnutí a vytvářejí statická rozhraní. GenUI naopak zahrnuje aktivnější roli AI systému v samotném generování a přizpůsobování rozhraní na základě uživatelských dat a kontextu.

GenUI představuje významný posun v tom, jak přistupujeme k návrhu uživatelského rozhraní, odklon od univerzálních řešení směrem k vysoce personalizovaným, adaptivním zážitkům. Využitím síly AI má GenUI potenciál revolucionizovat způsob, jakým interagujeme s digitálními produkty a službami, vytvářením rozhraní, která jsou intuitivnější, poutavější a efektivnější pro každého jednotlivého uživatele.

Příklad

Pro ilustraci konceptu GenUI uvažujme hypotetickou fitness aplikaci nazvanou "FitAI". Tato aplikace si klade za cíl poskytovat personalizované tréninkové plány a výživové rady uživatelům na základě jejich individuálních cílů, úrovně fyzické kondice a preferencí.

V tradičním přístupu k návrhu UI by FitAI mohla mít pevnou sadu obrazovek a prvků, které jsou stejné pro všechny uživatele. S GenUI by se však rozhraní aplikace mohlo dynamicky přizpůsobovat jedinečným potřebám a kontextu každého uživatele.

Tento přístup je v roce 2024 poměrně obtížné si představit implementovat a možná by ani neměl odpovídající návratnost investic, ale je možný.

Takto by to mohlo fungovat:

1. **Onboarding**:

    - Místo standardního dotazníku používá FitAI konverzační AI ke shromažďování informací o cílech uživatele, současné úrovni fyzické kondice a preferencích.

- Na základě této úvodní interakce AI generuje personalizované rozložení dashboardu, zvýrazňující funkce a informace nejrelevantnější pro cíle uživatele.
- Současná AI technologie by mohla mít k dispozici výběr komponent obrazovky pro sestavení personalizovaného dashboardu.
- Budoucí AI technologie by mohla převzít roli zkušeného UI designéra a skutečně vytvářet dashboard *od základu*.

2. **Plánovač tréninků**:

- Rozhraní plánovače tréninků je upravováno umělou inteligencí tak, aby přesně odpovídalo úrovni zkušeností uživatele a dostupnému vybavení.
- Pro začátečníka bez vybavení může zobrazovat jednoduché cviky s vlastní vahou těla s podrobnými instrukcemi a videi.
- Pro pokročilého uživatele s přístupem do posilovny může zobrazovat složitější rutiny s menším množstvím vysvětlujícího obsahu.
- Obsah plánovače tréninků není jen filtrován z velké nadmnožiny. Může být generován *za běhu* na základě znalostní báze, která je dotazována s kontextem zahrnujícím vše, co je o uživateli známo.

3. **Sledování pokroku**:

- Rozhraní pro sledování pokroku se vyvíjí na základě cílů uživatele a vzorců jeho zapojení.
- Pokud se uživatel primárně zaměřuje na hubnutí, rozhraní může výrazně zobrazovat graf trendu váhy a statistiky spalování kalorií.
- Pro uživatele budujícího svaly může zdůrazňovat nárůst síly a změny tělesné kompozice.
- UI může tuto část aplikace přizpůsobit skutečnému pokroku uživatele. Pokud se pokrok na určitou dobu zastaví, aplikace se může přepnout do režimu, kdy se snaží uživatele přimět k odhalení důvodů této překážky, aby je mohla zmírnit.

4. **Výživové poradenství**:
    - Sekce výživy se přizpůsobuje stravovacím preferencím a omezením uživatele.
    - Pro veganského uživatele může zobrazovat rostlinné návrhy jídel a zdroje bílkovin.
    - Pro uživatele s nesnášenlivostí lepku by automaticky filtrovalo potraviny obsahující lepek z doporučení.
    - I zde není obsah čerpán z masivní nadmnožiny dat o jídle, která platí pro všechny uživatele, ale je syntetizován ze znalostní báze obsahující informace přizpůsobitelné konkrétní situaci a omezením uživatele.
    - Například recepty jsou generovány se specifikacemi ingrediencí, které odpovídají neustále se měnícím kalorickým potřebám uživatele v závislosti na vývoji jeho fyzické kondice a tělesných statistik.
5. **Motivační prvky**:
    - Motivační obsah aplikace a notifikace jsou personalizovány na základě typu osobnosti uživatele a reakce na různé motivační strategie.
    - Někteří uživatelé mohou dostávat povzbuzující zprávy, zatímco jiní získávají více datově orientovanou zpětnou vazbu.

V tomto příkladu GenUI umožňuje aplikaci FitAI vytvořit vysoce přizpůsobenou zkušenost pro každého uživatele, potenciálně zvyšující zapojení, spokojenost a pravděpodobnost dosažení fitness cílů. Prvky rozhraní, obsah a dokonce i "osobnost" aplikace se přizpůsobují tak, aby co nejlépe sloužily potřebám a preferencím každého jednotlivého uživatele.

Posun k designu orientovanému na výsledky

GenUI představuje zásadní posun v přístupu k návrhu uživatelského rozhraní, přechod od zaměření na vytváření specifických prvků rozhraní k více holistickému přístupu orientovanému na výsledky. Tento posun má několik důležitých důsledků:

1. **Zaměření na cíle uživatelů**:

    - Designéři budou muset hlouběji přemýšlet o cílech uživatelů a požadovaných výsledcích spíše než o konkrétních komponentách rozhraní.
    - Důraz bude kladen na vytváření systémů, které mohou generovat rozhraní pomáhající uživatelům efektivně dosahovat jejich cílů.
    - Objeví se nové UI frameworky, které poskytnou AI-based designérům nástroje potřebné ke generování uživatelských zkušeností *za běhu* a *od základů* namísto předem definovaných specifikací obrazovek.

2. **Měnící se role designérů**:

    - Designéři přejdou od vytváření fixních layoutů k definování pravidel, omezení a pokynů pro AI systémy, které se jimi budou řídit při generování rozhraní.
    - Budou muset rozvíjet dovednosti v oblastech jako je analýza dat, inženýrství AI promptů a systémové myšlení, aby mohli efektivně vést GenUI systémy.

3. **Důležitost uživatelského výzkumu**:

    - Uživatelský výzkum se stává ještě kritičtějším v kontextu GenUI, protože designéři potřebují porozumět nejen preferencím uživatelů, ale také tomu, jak se tyto preference a potřeby mění v různých kontextech.
    - Kontinuální uživatelské testování a zpětnovazební smyčky budou zásadní pro zdokonalení a zlepšení schopnosti AI generovat efektivní rozhraní.

4. **Design pro variabilitu**:

    - Místo vytváření jediného "perfektního" rozhraní budou designéři muset zvažovat více možných variant a zajistit, že systém dokáže generovat vhodná rozhraní pro různé potřeby uživatelů.

- To zahrnuje design pro krajní případy a zajištění, že generovaná rozhraní zachovávají použitelnost a přístupnost napříč různými konfiguracemi.
- Diferenciace produktů získává nové dimenze zahrnující rozdílné pohledy na uživatelskou psychologii a využívání jedinečných datových sad a znalostních bází nedostupných konkurenci.

Výzvy a úvahy

Zatímco GenUI nabízí vzrušující možnosti, přináší také několik výzev a aspektů k zamyšlení:

1. **Technická omezení**:

    - Současná AI technologie, ačkoli pokročilá, má stále omezení v porozumění komplexním záměrům uživatelů a generování skutečně kontextově uvědomělých rozhraní.
    - Problémy s výkonem související s generováním prvků rozhraní v reálném čase, zejména na méně výkonných zařízeních.

2. **Požadavky na data**:

    - V závislosti na případu použití mohou efektivní systémy GenUI vyžadovat významné množství uživatelských dat pro generování personalizovaných rozhraní.
    - Výzvy v etickém získávání autentických uživatelských dat vyvolávají obavy ohledně ochrany osobních údajů a bezpečnosti, stejně jako potenciální předpojatosti v datech používaných k trénování modelů GenUI.

3. **Použitelnost a konzistence**:

- Přinejmenším dokud se tato praxe nestane běžnou, aplikace s neustále se měnícími rozhraními může vést k problémům s použitelností, protože uživatelé mohou mít potíže s nalezením známých prvků nebo efektivní navigací.
- Klíčové bude najít rovnováhu mezi personalizací a zachováním konzistentního, naučitelného rozhraní.

4. **Přílišné spoléhání na UI**:

   - Existuje riziko nadměrného delegování designových rozhodnutí na systémy UI, což může potenciálně vést k neinspiratívním, problematickým nebo jednoduše nefunkčním volbám rozhraní.
   - Lidský dohled a možnost přepsat AI generované návrhy zůstanou v dohledné budoucnosti důležité.

5. **Obavy ohledně přístupnosti**:

   - Zajištění, aby dynamicky generovaná rozhraní zůstala přístupná uživatelům s hendikepem, představuje zcela nové výzvy, což je znepokojující vzhledem k nízké úrovni dodržování přístupnosti u typických systémů.
   - Na druhou stranu, AI designéři mohou být implementováni s *vestavěnou* péčí o přístupnost a schopnostmi vytvářet přístupná rozhraní za běhu stejně jako vytvářejí UI pro uživatele bez hendikepů.
   - V každém případě by systémy GenUI měly být navrženy s robustními směrnicemi pro přístupnost a testovacími procesy.

6. **Důvěra uživatelů a transparentnost**:

   - Uživatelé se mohou cítit nepohodlně s rozhraními, která "vědí příliš mnoho" o nich nebo se mění způsoby, kterým nerozumí.
   - Pro budování důvěry uživatelů bude důležité poskytovat transparentnost ohledně toho, jak a proč jsou rozhraní personalizována.

Budoucí výhled a příležitosti

Budoucnost Generativního UI (GenUI) skýtá obrovský příslib pro revoluci ve způsobu, jakým interagujeme s digitálními produkty a službami. Jak se tato technologie nadále vyvíjí, můžeme očekávat zásadní změnu v tom, jak jsou uživatelská rozhraní navrhována, implementována a používána. Myslím, že GenUI je fenomén, který konečně posune náš software do oblasti toho, co je nyní považováno za vědeckou fantastiku.

Jednou z nejzajímavějších vyhlídek GenUI je jeho potenciál zlepšit přístupnost v měřítku, které jde nad rámec pouhého zajištění, aby lidé s vážným hendikepem nebyli zcela vyloučeni z používání vašeho softwaru. Automatickým přizpůsobováním rozhraní individuálním potřebám uživatelů by GenUI mohlo učinit digitální zkušenosti inkluzivnější než kdy předtím. Představte si rozhraní, která se plynule přizpůsobují tak, aby poskytovala větší text pro mladší nebo zrakově postižené uživatele nebo zjednodušená rozložení pro ty s kognitivními poruchami, to vše bez nutnosti manuální konfigurace nebo samostatných "přístupných" verzí aplikací.

Schopnosti personalizace GenUI pravděpodobně povedou ke zvýšení uživatelské angažovanosti, spokojenosti a loajality napříč širokou škálou digitálních produktů. Jak se rozhraní stávají více naladěná na individuální preference a chování, uživatelé budou považovat digitální zkušenosti za intuitivnější a příjemnější, což potenciálně povede k hlubším a smysluplnějším interakcím s technologií.

GenUI má také potenciál transformovat proces zaškolování nových uživatelů. Vytvářením intuitivních, personalizovaných zkušeností pro nové uživatele, které se rychle přizpůsobují úrovni odbornosti každého uživatele, by GenUI mohlo výrazně snížit křivku učení spojenou s novými aplikacemi. To by mohlo vést k rychlejším mírám osvojení a zvýšené důvěře uživatelů při zkoumání nových funkcí a funkcionalit.

Další vzrušující možností je schopnost GenUI udržovat konzistentní uživatelskou zkušenost napříč různými zařízeními a platformami při optimalizaci pro každý

specifický kontext použití. To by mohlo vyřešit dlouhodobou výzvu poskytování koherentních zkušeností napříč stále více fragmentovanou krajinou zařízení, od chytrých telefonů a tabletů po stolní počítače a vznikající technologie jako brýle pro rozšířenou realitu.

Datově řízená povaha GenUI otevírá příležitosti pro rychlou iteraci a zlepšování v návrhu UI. Shromažďováním dat v reálném čase o tom, jak uživatelé interagují s generovanými rozhraními, mohou designéři a vývojáři získat bezprecedentní vhled do uživatelského chování a preferencí. Tato zpětná vazba by mohla vést k neustálému zlepšování návrhu UI, řízenému skutečnými vzorci používání spíše než předpoklady nebo omezeným uživatelským testováním.

Pro přípravu na tuto změnu budou designéři muset rozvíjet své dovednosti a způsob myšlení. Zaměření se přesune od vytváření fixních layoutů k vývoji komplexních designových systémů a směrnic, které mohou informovat generování rozhraní řízené AI. Designéři budou muset rozvíjet hluboké porozumění datové analýze, AI technologiím a systémovému myšlení, aby efektivně vedli systémy GenUI.

Navíc, jak GenUI stírá hranice mezi designem a technologií, designéři budou muset úžeji spolupracovat s vývojáři a datovými vědci. Tento interdisciplinární přístup bude klíčový při vytváření systémů GenUI, které jsou nejen vizuálně přitažlivé a uživatelsky přívětivé, ale také technicky robustní a eticky zodpovědné.

Etické důsledky GenUI se dostanou do popředí s tím, jak bude technologie dozrávat. Designéři budou hrát klíčovou roli při vývoji rámců pro odpovědné využití AI v návrhu rozhraní, zajišťující, že personalizace zlepší uživatelské zkušenosti bez kompromitování soukromí či neetické manipulace s chováním uživatelů.

Při pohledu do budoucnosti představuje GenUI jak vzrušující příležitosti, tak významné výzvy. Má potenciál vytvářet intuitivnější, efektivnější a uspokojivější digitální zážitky pro uživatele po celém světě. Ačkoli to bude vyžadovat, aby si designéři osvojili nové dovednosti a přizpůsobili se, nabízí to také bezprecedentní příležitost formovat budoucnost interakce člověka s počítačem zásadním a smysluplným způsobem. Cesta

k plně realizovaným systémům GenUI bude bezpochyby složitá, ale potenciální přínosy v podobě vylepšených uživatelských zkušeností a digitální přístupnosti z ní činí budoucnost, o kterou stojí za to usilovat.

Inteligentní orchestrace pracovních postupů

V oblasti vývoje aplikací hrají pracovní postupy klíčovou roli při definování způsobu strukturování a provádění úkolů, procesů a uživatelských interakcí. S rostoucí složitostí aplikací a zvyšujícími se očekáváními uživatelů se stává stále zřejmější potřeba inteligentní a adaptivní orchestrace pracovních postupů.

Přístup "Inteligentní orchestrace pracovních postupů" se zaměřuje na využití komponent umělé inteligence k dynamické orchestraci a optimalizaci komplexních pracovních postupů v aplikacích. Cílem je vytvářet aplikace, které jsou efektivnější, responzivnější a přizpůsobivější vzhledem k datům a kontextu v reálném čase.

V této kapitole prozkoumáme klíčové principy a vzory, které tvoří základ přístupu inteligentní orchestrace pracovních postupů. Budeme se zabývat tím, jak lze využít

umělou inteligenci k inteligentnímů směrování úkolů, automatizaci rozhodování a dynamickému přizpůsobování pracovních postupů na základě různých faktorů, jako je chování uživatelů, výkon systému a obchodní pravidla. Prostřednictvím praktických příkladů a scénářů z reálného světa ukážeme transformační potenciál umělé inteligence při zefektivňování a optimalizaci pracovních postupů aplikací.

Ať už vytváříte podnikové aplikace se složitými obchodními procesy nebo aplikace zaměřené na spotřebitele s dynamickými uživatelskými cestami, vzory a techniky diskutované v této kapitole vám poskytnou znalosti a nástroje potřebné k vytváření inteligentních a efektivních pracovních postupů, které zlepšují celkový uživatelský zážitek a přinášejí obchodní hodnotu.

Obchodní potřeba

Tradiční přístupy k řízení pracovních postupů často spoléhají na předem definovaná pravidla a statické rozhodovací stromy, které mohou být rigidní, neflexibilní a neschopné vypořádat se s dynamickou povahou moderních aplikací.

Uvažujme scénář, kdy e-commerce aplikace potřebuje zpracovat komplexní proces vyřízení objednávky. Pracovní postup může zahrnovat několik kroků, jako je validace objednávky, kontrola skladu, zpracování platby, expedice a oznámení zákazníkům. Každý krok může mít vlastní sadu pravidel, závislostí, externí integrace a mechanismy pro zpracování výjimek. Ruční správa takového pracovního postupu nebo jeho řízení pomocí pevně nakódované logiky se může rychle stát těžkopádnou, náchylnou k chybám a obtížně udržovatelnou.

Navíc, jak aplikace roste a počet současně připojených uživatelů se zvyšuje, pracovní postup se může potřebovat přizpůsobovat a optimalizovat na základě dat v reálném čase a výkonu systému. Například během období špičkového provozu může aplikace potřebovat dynamicky upravit pracovní postup tak, aby upřednostnila určité úkoly, efektivně alokovala zdroje a zajistila plynulý uživatelský zážitek.

Zde přichází ke slovu přístup "Inteligentní orchestrace pracovních postupů". Využitím komponent umělé inteligence mohou vývojáři vytvářet pracovní postupy, které jsou inteligentní, adaptivní a samo-optimalizující. Umělá inteligence může analyzovat velké množství dat, učit se z minulých zkušeností a činit informovaná rozhodnutí v reálném čase pro efektivní orchestraci pracovního postupu.

Klíčové výhody

1. **Zvýšená efektivita**: Umělá inteligence může optimalizovat přidělování úkolů, využití zdrojů a provádění pracovních postupů, což vede k rychlejším dobám zpracování a zlepšené celkové efektivitě.
2. **Adaptabilita**: Pracovní postupy řízené umělou inteligencí se mohou dynamicky přizpůsobovat měnícím se podmínkám, jako jsou výkyvy v poptávce uživatelů, výkonu systému nebo obchodních požadavcích, což zajišťuje, že aplikace zůstává responzivní a odolná.
3. **Automatizované rozhodování**: Umělá inteligence může automatizovat složité rozhodovací procesy v rámci pracovního postupu, čímž snižuje potřebu manuálních zásahů a minimalizuje riziko lidských chyb.
4. **Personalizace**: Umělá inteligence může analyzovat chování uživatelů, preference a kontext pro personalizaci pracovního postupu a poskytování přizpůsobených zážitků jednotlivým uživatelům.
5. **Škálovatelnost**: Pracovní postupy poháněné umělou inteligencí se mohou plynule škálovat pro zvládání rostoucího objemu dat a uživatelských interakcí, aniž by byl ohrožen výkon nebo spolehlivost.

V následujících částech prozkoumáme klíčové vzory a techniky, které umožňují implementaci inteligentních pracovních postupů, a ukážeme příklady z reálného světa, jak umělá inteligence transformuje řízení pracovních postupů v moderních aplikacích.

Klíčové vzory

Pro implementaci inteligentní orchestrace pracovních postupů v aplikacích mohou vývojáři využít několik klíčových vzorů, které využívají sílu umělé inteligence. Tyto vzory poskytují strukturovaný přístup k návrhu a řízení pracovních postupů, umožňující aplikacím přizpůsobovat se, optimalizovat a automatizovat procesy na základě dat a kontextu v reálném čase. Pojďme prozkoumat některé ze základních vzorů v inteligentní orchestraci pracovních postupů.

Dynamické směrování úloh

Tento vzor zahrnuje využití umělé inteligence k inteligentnímu směrování úloh v rámci pracovního postupu na základě různých faktorů, jako je priorita úlohy, dostupnost zdrojů a výkon systému. Algoritmy umělé inteligence mohou analyzovat charakteristiky každé úlohy, zvážit aktuální stav systému a činit informovaná rozhodnutí pro přiřazení úloh nejvhodnějším zdrojům nebo cestám zpracování. Dynamické směrování úloh zajišťuje efektivní distribuci a provádění úloh, optimalizující celkový výkon pracovního postupu.

```
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
```

Všimněte si smyčky vytvořené výrazem `while` na řádku 29, která pokračuje v dotazování AI, dokud není úkol přiřazen. Na řádku 35 ukládáme přepis úkolu pro pozdější analýzu a ladění, pokud to bude nutné.

Kontextové rozhodování

Můžete použít velmi podobný kód k vytváření kontextově uvědomělých rozhodnutí v rámci pracovního postupu. Analyzováním relevantních datových bodů, jako jsou uživatelské preference, historické vzory a vstupy v reálném čase, mohou AI komponenty určit nejvhodnější postup v každém rozhodovacím bodě pracovního postupu. Přizpůsobte chování vašeho pracovního postupu na základě specifického kontextu každého uživatele nebo scénáře, poskytujíce personalizované a optimalizované zkušenosti.

Adaptivní kompozice pracovních postupů

Tento vzor se zaměřuje na dynamické sestavování a úpravu pracovních postupů na základě měnících se požadavků nebo podmínek. AI může analyzovat současný stav pracovního postupu, identifikovat úzká místa nebo neefektivity a automaticky upravit strukturu pracovního postupu pro optimalizaci výkonu. Adaptivní kompozice pracovních postupů umožňuje aplikacím neustále se vyvíjet a zlepšovat své procesy bez nutnosti manuálního zásahu.

Zpracování a zotavení z výjimek

Zpracování a zotavení z výjimek jsou kritické aspekty inteligentní orchestrace pracovních postupů. Při práci s AI komponentami a komplexními pracovními postupy je zásadní předvídat a elegantně zpracovávat výjimky pro zajištění stability a spolehlivosti systému.

Zde jsou klíčové úvahy a techniky pro zpracování a zotavení z výjimek v inteligentních pracovních postupech:

1. **Propagace výjimek**: Implementujte konzistentní přístup pro propagaci výjimek napříč komponentami pracovního postupu. Když dojde k výjimce uvnitř komponenty, měla by být zachycena, zaznamenána a propagována do orchestrátoru nebo samostatné komponenty zodpovědné za zpracování výjimek. Myšlenkou je centralizovat zpracování výjimek a zabránit tichému pohlcování výjimek, stejně jako otevřít možnosti pro Inteligentní zpracování chyb.
2. **Mechanismy opakování**: Mechanismy opakování pomáhají zlepšit odolnost pracovního postupu a elegantně zvládat přechodná selhání. Rozhodně se snažte implementovat mechanismy opakování pro přechodné nebo obnovitelné výjimky, jako je síťové připojení nebo nedostupnost zdrojů, které lze automaticky znovu zkusit po stanovené prodlevě. Mít AI-řízený orchestrátor nebo zpracovatel

výjimek znamená, že vaše strategie opakování nemusí být mechanické povahy, spoléhající se na pevné algoritmy jako exponenciální odstup. Můžete ponechat zpracování opakování na "uvážení" AI komponenty zodpovědné za rozhodování o tom, jak výjimku zpracovat.

3. **Záložní strategie**: Pokud AI komponenta selže v poskytnutí platné odpovědi nebo narazí na chybu—běžný jev vzhledem k její průkopnické povaze—mějte připraven záložní mechanismus, který zajistí pokračování pracovního postupu. To může zahrnovat použití výchozích hodnot, alternativních algoritmů nebo Člověka v procesu pro rozhodování a udržení pracovního postupu v chodu.
4. **Kompenzační akce**: Pokyny orchestrátoru by měly zahrnovat instrukce o kompenzačních akcích pro zpracování výjimek, které nelze vyřešit automaticky. Kompenzační akce jsou kroky podniknuté k vrácení nebo zmírnění účinků neúspěšné operace. Například pokud selže krok zpracování platby, kompenzační akce by mohla být vrácení transakce a upozornění uživatele. Kompenzační akce pomáhají udržovat konzistenci dat a integritu v případě výjimek.
5. **Monitorování a upozorňování na výjimky**: Nastavte mechanismy monitorování a upozorňování pro detekci a oznámení relevantním zainteresovaným stranám o kritických výjimkách. Orchestrátor může být informován o prahových hodnotách a pravidlech pro spouštění upozornění, když výjimky překročí určité limity nebo když dojde ke specifickým typům výjimek. To umožňuje proaktivní identifikaci a řešení problémů předtím, než ovlivní celkový systém.

Zde je příklad zpracování a zotavení z výjimek v komponentě pracovního postupu v 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
```

V tomto příkladu komponenta `InventoryManager` kontroluje dostupnost produktu pro danou objednávku. Pokud je dostupné množství nedostatečné, vyvolá `InsufficientInventoryError`. Výjimka je zachycena, zaznamenána a je implementován mechanismus opakování. Pokud je překročen limit opakování, komponenta přejde k manuálnímu zásahu tím, že upozorní administrátora.

Implementací robustního zpracování výjimek a mechanismů obnovy můžete zajistit, že vaše inteligentní workflow budou odolné, udržovatelné a schopné elegantně zvládat

neočekávané situace.

Tyto vzory tvoří základ inteligentní orchestrace workflow a lze je kombinovat a přizpůsobovat specifickým požadavkům různých aplikací. Využitím těchto vzorů mohou vývojáři vytvářet workflow, které jsou flexibilní, odolné a optimalizované pro výkon a uživatelskou zkušenost.

V další části prozkoumáme, jak lze tyto vzory implementovat v praxi, s využitím příkladů z reálného světa a ukázek kódu pro ilustraci integrace AI komponent do řízení workflow.

Implementace inteligentní orchestrace workflow v praxi

Nyní, když jsme prozkoumali klíčové vzory v inteligentní orchestraci workflow, pojďme se ponořit do toho, jak lze tyto vzory implementovat v reálných aplikacích. Poskytneme praktické příklady a ukázky kódu pro ilustraci integrace AI komponent do řízení workflow.

Inteligentní zpracování objednávek

Pojďme se ponořit do praktického příkladu implementace inteligentní orchestrace workflow pomocí AI komponenty `OrderProcessor` v e-commerce aplikaci Ruby on Rails. `OrderProcessor` realizuje koncept Process Manager Enterprise Integration, se kterým jsme se poprvé setkali v Kapitole 3 při diskuzi o Množství pracovníků. Komponenta bude zodpovědná za řízení workflow vyřizování objednávek, rozhodování o směrování na základě průběžných výsledků a orchestraci provádění různých kroků zpracování.

Proces vyřizování objednávek zahrnuje několik kroků, jako je validace objednávky, kontrola zásob, zpracování platby a expedice. Každý krok je implementován jako samostatný pracovní proces, který provádí specifický úkol a vrací výsledek zpět do `OrderProcessor`. Kroky nejsou povinné a dokonce nemusí být nutně provedeny v přesném pořadí.

Zde je příklad implementace `OrderProcessor`. Obsahuje dva mixiny z Raix. První (`ChatCompletion`) mu dává schopnost dokončování chatu, což z něj dělá AI komponentu. Druhý (`FunctionDispatch`) umožňuje volání funkcí umělou inteligencí, což jí dovoluje reagovat na prompt voláním funkce místo textové zprávy.

Pracovní funkce (`validate_order`, `check_inventory`, atd.) delegují na své příslušné pracovní třídy, které mohou být AI nebo ne-AI komponenty, s jediným požadavkem, že musí vracet výsledky své práce ve formátu, který lze reprezentovat jako řetězec.

Stejně jako u všech ostatních příkladů v této části knihy je tento kód prakticky pseudokódem a má pouze zprostředkovat význam vzoru a inspirovat vaše vlastní výtvory. Úplné popisy vzorů a kompletní příklady kódu jsou zahrnuty v Části 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)
```

```
16    end
17
18    # list of functions available to be called by the AI
19    # truncated for brevity
20
21    def functions
22      [
23        {
24          name: "validate_order",
25          description: "Invoke to check validity of order",
26          parameters: {
27            ...
28        },
29        ...
30      ]
31    end
32
33    # implementation of functions that can be called by the AI
34    # entirely at its discretion, depending on the needs of the order
35
36    def validate_order
37      OrderValidationWorker.perform(@order)
38    end
39
40    def check_inventory
41      InventoryCheckWorker.perform(@order)
42    end
43
44    def process_payment
45      PaymentProcessingWorker.perform(@order)
46    end
47
48    def schedule_shipping
49      ShippingSchedulerWorker.perform(@order)
50    end
51
52    def send_confirmation
53      OrderConfirmationWorker.perform(@order)
54    end
55
56    def finished_processing
57      @order.update!(transcript:, processed_at: Time.current)
```

```
    stop_looping!
  end
end
```

V uvedeném příkladu je OrderProcessor inicializován s objektem objednávky a udržuje přepis průběhu pracovního postupu ve formátu konverzačního přepisu, který je přirozený pro velké jazykové modely. AI dostává plnou kontrolu nad orchestrací provádění různých kroků zpracování, jako je validace objednávky, kontrola zásob, zpracování plateb a expedice.

Pokaždé, když je volána metoda `chat_completion`, je přepis odeslán AI, aby poskytla dokončení ve formě volání funkce. Je zcela na AI, aby analyzovala výsledek předchozího kroku a určila vhodnou akci. Například pokud kontrola zásob odhalí nízký stav zásob, může `OrderProcessor` naplánovat úkol doplnění. Pokud selže zpracování platby, může iniciovat opakovaný pokus nebo upozornit zákaznickou podporu.

> Výše uvedený příklad nemá definované funkce pro doplňování zásob nebo upozorňování zákaznické podpory, ale rozhodně by je mít mohl.

Přepis roste s každým voláním funkce a slouží jako záznam o průběhu pracovního postupu, včetně výsledků každého kroku a AI generovaných instrukcí pro další kroky. Tento přepis lze použít pro ladění, audit a poskytování přehledu o procesu vyřizování objednávek.

Využitím AI v `OrderProcessor` se může e-commerce aplikace dynamicky přizpůsobovat pracovnímu postupu na základě dat v reálném čase a inteligentně zvládat výjimky. AI komponenta může činit informovaná rozhodnutí, optimalizovat pracovní postup a zajistit plynulé zpracování objednávek i ve složitých scénářích.

Skutečnost, že jediným požadavkem na pracovní procesy je vrátit nějaký srozumitelný výstup, který AI zváží při rozhodování o dalším postupu, vám může začít naznačovat,

jak tento přístup může snížit práci spojenou s mapováním vstupů a výstupů, která je typicky nutná při integraci různorodých systémů.

Inteligentní moderátor obsahu

Aplikace sociálních médií obecně vyžadují alespoň minimální moderování obsahu pro zajištění bezpečné a zdravé komunity. Tento příklad komponenty `ContentModerator` využívá AI k inteligentní orchestraci moderačního workflow, přičemž rozhodnutí jsou založena na charakteristikách obsahu a výsledcích různých moderačních kroků.

Moderační proces zahrnuje více kroků, jako je analýza textu, rozpoznávání obrázků, hodnocení reputace uživatele a manuální kontrola. Každý krok je implementován jako samostatný pracovní proces, který provádí specifický úkol a vrací výsledek do `ContentModerator`.

Zde je příklad implementace `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
```

```
64
65    def recognize_image
66      result = ImageRecognitionWorker.perform(@content)
67      continue_with(result)
68    end
69
70    def assess_user_reputation
71      result = UserReputationWorker.perform(@content.user)
72      continue_with(result)
73    end
74
75    def escalate_to_manual_review
76      ManualReviewWorker.perform(@content)
77      @content.update!(status: 'pending', transcript: @transcript)
78    end
79
80    def approve_content
81      @content.update!(status: 'approved', transcript: @transcript)
82    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
```

V tomto příkladu je `ContentModerator` inicializován s objektem obsahu a udržuje moderační záznam v konverzačním formátu. AI komponenta má plnou kontrolu nad moderačním postupem a rozhoduje, které kroky provést na základě charakteristik obsahu a výsledků každého kroku.

Dostupné pracovní funkce, které může AI vyvolat, zahrnují `analyze_text`, `recognize_image`, `assess_user_reputation` a `escalate_to_manual_review`. Každá funkce deleguje úkol na odpovídající pracovní proces (`TextAnalysisWorker`,

`ImageRecognitionWorker`, atd.) a připojuje výsledek do moderačního záznamu, s výjimkou funkce eskalace, která působí jako koncový stav. Nakonec funkce `approve_content` a `reject_content` také působí jako koncové stavy.

AI komponenta analyzuje obsah a určuje vhodnou akci. Pokud obsah obsahuje odkazy na obrázky, může pro pomoc s vizuální kontrolou zavolat pracovní funkci `recognize_image`. Pokud některý pracovní proces upozorní na potenciálně škodlivý obsah, AI se může rozhodnout eskalovat obsah k manuální kontrole nebo jej rovnou zamítnout. Ale v závislosti na závažnosti varování se AI může rozhodnout využít výsledky hodnocení reputace uživatele při rozhodování, jak naložit s obsahem, u kterého si není jinak jistá. V závislosti na případu použití mohou mít například důvěryhodní uživatelé větší volnost v tom, co mohou zveřejnit. A tak dále a tak podobně…

Stejně jako v předchozím příkladu správce procesů slouží moderační záznam jako evidence provedení pracovního postupu, včetně výsledků každého kroku a rozhodnutí generovaných AI. Tento záznam lze využít pro audit, transparentnost a zlepšování moderačního procesu v průběhu času.

Využitím AI v `ContentModerator` může aplikace sociálních médií dynamicky přizpůsobovat moderační postup na základě charakteristik obsahu a inteligentně zvládat komplexní moderační scénáře. AI komponenta může činit informovaná rozhodnutí, optimalizovat pracovní postup a zajistit bezpečnou a zdravou komunitní zkušenost.

Prozkoumejme další dva příklady, které demonstrují prediktivní plánování úloh a zpracování výjimek a zotavení v kontextu inteligentní orchestrace pracovního postupu.

Prediktivní plánování úloh v systému zákaznické podpory

V aplikaci zákaznické podpory vytvořené pomocí Ruby on Rails je efektivní správa a prioritizace požadavků podpory klíčová pro poskytování včasné pomoci zákazníkům.

Komponenta `SupportTicketScheduler` využívá AI k prediktivnímu plánování a přiřazování požadavků podpory dostupným agentům na základě různých faktorů, jako je naléhavost požadavku, odbornost agenta a pracovní vytížení.

```
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
```

V tomto příkladu je `SupportTicketScheduler` inicializován s objektem požadavku podpory a udržuje záznam plánování. Komponenta AI analyzuje detaily požadavku a prediktivně plánuje jeho přiřazení na základě faktorů jako je naléhavost požadavku, odbornost agenta a předpokládané pracovní zatížení agenta.

Dostupné funkce, které může AI vyvolat, zahrnují `analyze_ticket_urgency`, `list_available_agents`, `predict_agent_workload` a `assign_ticket_to_agent`. Každá funkce deleguje úkol na příslušnou analyzační nebo predikční komponentu a připojuje výsledek k záznamu plánování. AI má také možnost odložit přiřazení pomocí funkce `delay_assignment`.

Komponenta AI zkoumá záznam plánování a činí informovaná rozhodnutí o přiřazení požadavků. Bere v úvahu naléhavost požadavku, odbornost dostupných agentů a předpokládané pracovní zatížení každého agenta, aby určila nejvhodnějšího agenta pro zpracování požadavku.

Využitím prediktivního plánování úkolů může aplikace zákaznické podpory optimalizovat přiřazování požadavků, zkrátit dobu odezvy a zlepšit celkovou spokojenost zákazníků. Proaktivní a efektivní správa požadavků podpory zajišťuje, že správné požadavky jsou přiřazeny správným agentům ve správný čas.

Zpracování výjimek a obnova v pipeline zpracování dat

Zpracování výjimek a obnova po selháních jsou nezbytné pro zajištění integrity dat a prevenci jejich ztráty. Komponenta `DataProcessingOrchestrator` využívá AI k inteligentnímu zpracování výjimek a orchestraci procesu obnovy v pipeline zpracování dat

```
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
```

V tomto příkladu je `DataProcessingOrchestrator` inicializován s objektem dávky dat a udržuje záznam o zpracování. AI komponenta orchestruje pipeline zpracování dat, řeší výjimky a zotavuje se z chyb podle potřeby.

Dostupné funkce, které může AI volat, zahrnují `validate_data`, `process_data`, `request_fix`, `retry_processing` a `mark_data_as_failed`. Každá funkce deleguje úkol na odpovídající komponentu zpracování dat a připojuje výsledek nebo podrobnosti o výjimce do záznamu o zpracování.

Pokud během kroku `validate_data` dojde k výjimce při validaci, funkce `handle_validation_exception` připojí data o výjimce do záznamu a předá řízení zpět AI. Podobně, pokud během kroku `process_data` dojde k výjimce při zpracování, AI může rozhodnout o strategii zotavení.

V závislosti na povaze vzniklé výjimky může AI podle svého uvážení rozhodnout o volání `request_fix`, které deleguje na AI komponentu `SmartDataFixer` (viz kapitola Samouzdravující se data). Opravný nástroj dat dostane v běžné angličtině

popis toho, jak by měl upravit `@data_batch`, aby bylo možné zpracování opakovat. Možná by úspěšné opakování znamenalo odstranění záznamů z dávky dat, které neprošly validací, a/nebo jejich zkopírování do jiné pipeline zpracování pro lidskou kontrolu? Možnosti jsou téměř nekonečné.

Začleněním zpracování výjimek a zotavení řízeného AI se aplikace pro zpracování dat stává odolnější a tolerantnější k chybám. `DataProcessingOrchestrator` inteligentně spravuje výjimky, minimalizuje ztrátu dat a zajišťuje plynulé provedení workflow zpracování dat.

Monitorování a protokolování

Monitorování a protokolování poskytují přehled o průběhu, výkonu a stavu komponent workflow řízených AI, což vývojářům umožňuje sledovat a analyzovat chování systému. Implementace efektivních mechanismů monitorování a protokolování je nezbytná pro ladění, audit a neustálé zlepšování inteligentních workflow.

Monitorování průběhu a výkonu workflow

Pro zajištění plynulého provádění inteligentních workflow je důležité sledovat průběh a výkon každé komponenty workflow. To zahrnuje sledování klíčových metrik a událostí během životního cyklu workflow.

Důležité aspekty ke sledování zahrnují:

1. **Doba provádění workflow**: Měření času, který každá komponenta workflow potřebuje k dokončení svého úkolu. To pomáhá identifikovat výkonnostní úzká místa a optimalizovat celkovou efektivitu workflow.

2. **Využití zdrojů**: Sledování využití systémových zdrojů, jako jsou CPU, paměť a úložiště, každou komponentou workflow. To pomáhá zajistit, že systém pracuje v rámci své kapacity a může efektivně zvládat pracovní zátěž.

3. **Míry chyb a výjimky**: Sledování výskytu chyb a výjimek v komponentách workflow. To pomáhá identifikovat potenciální problémy a umožňuje proaktivní zpracování a zotavení z chyb.

4. **Rozhodovací body a výsledky**: Sledování rozhodovacích bodů v rámci workflow a výsledků rozhodnutí řízených AI. To poskytuje vhled do chování a efektivity AI komponent.

Data zachycená monitorovacími procesy mohou být zobrazena v dashboardech nebo použita jako vstupy pro plánované zprávy, které informují správce systému o jeho stavu.

Monitorovací data mohou být předána procesu správce systému řízenému AI ke kontrole a případné akci!

Protokolování klíčových událostí a rozhodnutí

Protokolování je zásadní praxe, která zahrnuje zachycování a ukládání relevantních informací o klíčových událostech, rozhodnutích a výjimkách, ke kterým dochází během provádění workflow.

Důležité aspekty k protokolování zahrnují:

1. **Zahájení a dokončení workflow**: Zaznamenávání času začátku a konce každé instance workflow, spolu s relevantními metadaty, jako jsou vstupní data a uživatelský kontext.

2. **Provádění komponent**: Zaznamenávání podrobností o provádění každé komponenty workflow, včetně vstupních parametrů, výstupních výsledků a veškerých vygenerovaných mezilehlých dat.

3. **Rozhodnutí AI a zdůvodnění**: Zaznamenávání rozhodnutí učiněných AI komponentami, spolu s podkladovým zdůvodněním nebo skóre spolehlivosti. To poskytuje transparentnost a umožňuje audit rozhodnutí řízených AI.

4. **Výjimky a chybové zprávy**: Zaznamenávání všech výjimek nebo chybových zpráv, se kterými se během provádění workflow setkáme, včetně zásobníkového výpisu a relevantních kontextových informací.

Protokolování lze implementovat pomocí různých technik, jako je zápis do souborů protokolu, ukládání protokolů v databázi nebo odesílání protokolů do centralizované služby protokolování. Je důležité zvolit framework pro protokolování, který poskytuje flexibilitu, škálovatelnost a snadnou integraci s architekturou aplikace.

Zde je příklad, jak lze implementovat protokolování v aplikaci Ruby on Rails pomocí třídy `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)
```

Strategickým umístěním protokolovacích záznamů v rámci komponent pracovních postupů a rozhodovacích bodů umělé inteligence mohou vývojáři získat cenné informace pro ladění, audit a analýzu.

Výhody monitorování a protokolování

Implementace monitorování a protokolování v inteligentní orchestraci pracovních postupů přináší několik výhod:

1. **Ladění a řešení problémů**: Podrobné protokoly a monitorovací data pomáhají vývojářům rychle identifikovat a diagnostikovat problémy. Poskytují přehled o průběhu vykonávání pracovního postupu, interakcích mezi komponenty a případných chybách či výjimkách.

2. **Optimalizace výkonu**: Monitorování výkonnostních metrik umožňuje vývojářům identifikovat úzká místa a optimalizovat komponenty pracovního postupu pro lepší efektivitu. Analýzou doby vykonávání, využití zdrojů a dalších metrik mohou vývojáři činit informovaná rozhodnutí pro zlepšení celkového výkonu systému.

3. **Audit a dodržování předpisů**: Protokolování klíčových událostí a rozhodnutí poskytuje auditní stopu pro regulační shodu a odpovědnost. Umožňuje organizacím sledovat a ověřovat činnosti prováděné komponenty umělé inteligence a zajistit dodržování obchodních pravidel a právních požadavků.

4. **Neustálé zlepšování**: Data z monitorování a protokolování slouží jako cenné vstupy pro neustálé zlepšování inteligentních pracovních postupů. Analýzou historických dat, identifikací vzorců a měřením efektivity rozhodnutí umělé inteligence mohou vývojáři iterativně zdokonalovat a vylepšovat logiku orchestrace pracovních postupů.

Úvahy a osvědčené postupy

Při implementaci monitorování a protokolování v inteligentní orchestraci pracovních postupů zvažte následující osvědčené postupy:

1. **Definujte jasné monitorovací metriky**: Identifikujte klíčové metriky a události, které je třeba monitorovat na základě specifických požadavků pracovního postupu. Zaměřte se na metriky, které poskytují smysluplné informace o výkonu, zdraví a chování systému.

2. **Implementujte podrobné protokolování**: Zajistěte, aby byly protokolovací záznamy umístěny na vhodných místech v rámci komponent pracovního postupu a rozhodovacích bodů umělé inteligence. Zachyťte relevantní kontextové informace,

jako jsou vstupní parametry, výstupní výsledky a veškerá vygenerovaná mezilehlá data.

3. **Používejte strukturované protokolování**: Používejte strukturovaný formát protokolování pro usnadnění snadného parsování a analýzy protokolovaných dat. Strukturované protokolování umožňuje lepší vyhledávání, filtrování a agregaci protokolových záznamů.

4. **Spravujte uchovávání a rotaci protokolů**: Implementujte zásady pro uchovávání a rotaci protokolů pro správu úložiště a životního cyklu protokolových souborů. Určete vhodnou dobu uchovávání na základě právních požadavků, omezení úložiště a potřeb analýzy. Pokud je to možné, přesuňte protokolování na službu třetí strany, jako je Papertrail.

5. **Zabezpečte citlivé informace**: Buďte opatrní při protokolování citlivých informací, jako jsou osobní údaje (PII) nebo důvěrné obchodní údaje. Implementujte vhodná bezpečnostní opatření, jako je maskování dat nebo šifrování, pro ochranu citlivých informací v protokolových souborech.

6. **Integrujte s monitorovacími a výstražnými nástroji**: Využijte monitorovací a výstražné nástroje pro centralizaci sběru, analýzy a vizualizace monitorovacích a protokolovacích dat. Tyto nástroje mohou poskytovat informace v reálném čase, generovat upozornění na základě předem definovaných prahových hodnot a usnadnit proaktivní detekci a řešení problémů. Mým oblíbeným z těchto nástrojů je Datadog.

Implementací komplexních mechanismů monitorování a protokolování mohou vývojáři získat cenné informace o chování a výkonu inteligentních pracovních postupů. Tyto poznatky umožňují efektivní ladění, optimalizaci a neustálé zlepšování systémů orchestrace pracovních postupů založených na umělé inteligenci.

Úvahy o škálovatelnosti a výkonu

Škálovatelnost a výkon jsou kritické aspekty, které je třeba zvážit při návrhu a implementaci systémů inteligentní orchestrace pracovních postupů. S rostoucím objemem souběžných pracovních postupů a složitostí komponent založených na umělé inteligenci je nezbytné zajistit, aby systém dokázal efektivně zvládat pracovní zátěž a bezproblémově se škálovat podle rostoucích požadavků.

Zvládání velkých objemů souběžných pracovních postupů

Systémy inteligentní orchestrace pracovních postupů často musí zvládat velké množství souběžných pracovních postupů. Pro zajištění škálovatelnosti zvažte následující strategie:

1. **Asynchronní zpracování**: Implementujte mechanismy asynchronního zpracování pro oddělení vykonávání komponent pracovního postupu. To umožňuje systému zpracovávat více pracovních postupů současně bez blokování nebo čekání na dokončení každé komponenty. Asynchronního zpracování lze dosáhnout pomocí front zpráv, architektur řízených událostmi nebo frameworků pro zpracování úloh na pozadí, jako je Sidekiq.

2. **Distribuovaná architektura**: Navrhněte architekturu systému tak, aby využívala bezserverové komponenty (jako je AWS Lambda) nebo jednoduše distribuovala pracovní zátěž mezi více uzlů či serverů spolu s vaším hlavním aplikačním serverem. To umožňuje horizontální škálovatelnost, kdy lze přidat další uzly pro zvládnutí zvýšených objemů pracovních postupů.

3. **Paralelní vykonávání**: Identifikujte příležitosti pro paralelní vykonávání v rámci pracovních postupů. Některé komponenty pracovního postupu mohou být na sobě nezávislé a lze je vykonávat současně. Využitím technik paralelního zpracování, jako je vícevláknové zpracování nebo distribuované fronty úloh, může systém optimalizovat využití zdrojů a zkrátit celkovou dobu vykonávání pracovního postupu.

Optimalizace výkonu komponent založených na umělé inteligenci

Komponenty založené na umělé inteligenci, jako jsou modely strojového učení nebo systémy pro zpracování přirozeného jazyka, mohou být výpočetně náročné a ovlivnit celkový výkon systému pro orchestraci pracovních postupů. Pro optimalizaci výkonu AI komponent zvažte následující techniky:

1. **Ukládání do mezipaměti:** Pokud je vaše AI zpracování čistě generativní a nezahrnuje vyhledávání informací v reálném čase nebo externí integrace pro generování chatových odpovědí, můžete se zaměřit na mechanismy ukládání do mezipaměti pro ukládání a opětovné použití výsledků často přistupovaných nebo výpočetně náročných operací.

2. **Optimalizace modelu**: Průběžně optimalizujte způsob, jakým používáte AI modely v komponentách pracovního postupu. To může zahrnovat techniky jako *Destilace promptů* nebo to může být jednoduše otázka testování nových modelů, když se stanou dostupnými.

3. **Dávkové zpracování**: Pokud pracujete s modely třídy GPT--4, můžete využít techniky dávkového zpracování pro zpracování více datových bodů nebo požadavků v jedné dávce, namísto jejich individuálního zpracování. Zpracováním dat v dávkách může systém optimalizovat využití zdrojů a snížit režii opakovaných požadavků na model.

Monitorování a profilování výkonu

Pro identifikaci výkonnostních úzkých míst a optimalizaci škálovatelnosti systému inteligentní orchestrace pracovních postupů je klíčové implementovat mechanismy monitorování a profilování. Zvažte následující přístupy:

1. **Metriky výkonu**: Definujte a sledujte klíčové metriky výkonu, jako je doba odezvy, propustnost, využití zdrojů a latence. Tyto metriky poskytují přehled o výkonu systému a pomáhají identifikovat oblasti pro optimalizaci. Populární agregátor AI modelů

OpenRouter zahrnuje metriky Host[1] a Speed[2] v každé API odpovědi, což usnadňuje sledování těchto klíčových metrik.

2. **Profilovací nástroje**: Využívejte profilovací nástroje k analýze výkonu jednotlivých komponent pracovního postupu a AI operací. Profilovací nástroje mohou pomoci identifikovat výkonnostní hotspoty, neefektivní cesty v kódu nebo operace náročné na zdroje. Mezi populární profilovací nástroje patří New Relic, Scout nebo vestavěné profilery poskytované programovacím jazykem nebo frameworkem.

3. **Zátěžové testování**: Provádějte zátěžové testování pro vyhodnocení výkonu systému při různých úrovních souběžného zatížení. Zátěžové testování pomáhá identifikovat limity škálovatelnosti systému, detekovat degradaci výkonu a zajistit, že systém zvládne očekávaný provoz bez kompromisů ve výkonu.

4. **Kontinuální monitoring**: Implementujte mechanismy kontinuálního monitorování a upozorňování pro proaktivní detekci problémů s výkonem a úzkých míst. Nastavte monitorovací dashboardy a upozornění pro sledování klíčových ukazatelů výkonu (KPI) a přijímání oznámení při překročení předem definovaných prahových hodnot. To umožňuje rychlou identifikaci a řešení problémů s výkonem.

Strategie škálování

Pro zvládnutí rostoucího zatížení a zajištění škálovatelnosti systému inteligentní orchestrace pracovních postupů zvažte následující strategie škálování:

1. **Vertikální škálování**: Vertikální škálování zahrnuje zvyšování zdrojů (např. CPU, paměti) jednotlivých uzlů nebo serverů pro zvládnutí vyššího zatížení. Tento přístup je vhodný, když systém vyžaduje více výpočetního výkonu nebo paměti pro zpracování komplexních pracovních postupů nebo AI operací.

[1] Host je čas, který byl potřeba k přijetí prvního bajtu streamovaného generování od hostitele modelu, také známý jako "čas do prvního bajtu."

[2] Speed se vypočítává jako počet dokončovacích tokenů dělený celkovým časem generování. Pro nestreamované požadavky se latence považuje za součást času generování.

2. **Horizontální škálování**: Horizontální škálování zahrnuje přidávání více uzlů nebo serverů do systému pro distribuci zátěže. Tento přístup je efektivní, když systém potřebuje zvládnout velký počet souběžných pracovních postupů nebo když lze zátěž snadno distribuovat mezi více uzlů. Horizontální škálování vyžaduje distribuovanou architekturu a mechanismy vyvažování zátěže pro zajištění rovnoměrné distribuce provozu.

3. **Automatické škálování**: Implementujte mechanismy automatického škálování pro automatické upravování počtu uzlů nebo zdrojů na základě požadavků na zátěž. Automatické škálování umožňuje systému dynamicky škálovat nahoru nebo dolů v závislosti na příchozím provozu, zajišťující optimální využití zdrojů a nákladovou efektivitu. Cloudové platformy jako Amazon Web Services (AWS) nebo Google Cloud Platform (GCP) poskytují možnosti automatického škálování, které lze využít pro systémy inteligentní orchestrace pracovních postupů.

Techniky optimalizace výkonu

Kromě strategií škálování zvažte následující techniky optimalizace výkonu pro zvýšení efektivity systému inteligentní orchestrace pracovních postupů:

1. **Efektivní ukládání a načítání dat**: Optimalizujte mechanismy ukládání a načítání dat používané komponentami pracovního postupu. Používejte efektivní indexování databáze, techniky optimalizace dotazů a ukládání dat do mezipaměti pro minimalizaci latence a zlepšení výkonu operací náročných na data.

2. **Asynchronní I/O**: Využijte asynchronní I/O operace k zabránění blokování a zlepšení odezvy systému. Asynchronní I/O umožňuje systému zpracovávat více požadavků současně bez čekání na dokončení I/O operací, čímž maximalizuje využití zdrojů.

3. **Efektivní serializace a deserializace**: Optimalizujte procesy serializace a deserializace používané pro výměnu dat mezi komponenty workflow. Používejte efektivní serializační formáty, jako jsou Protocol Buffers nebo MessagePack, ke snížení režie datové serializace a zlepšení výkonu komunikace mezi komponenty.

Pro aplikace založené na Ruby zvažte použití Universal ID. Universal ID využívá MessagePack i Brotli (kombinaci vytvořenou pro rychlost a špičkovou kompresi dat). Při společném použití jsou tyto knihovny až o 30 % rychlejší a dosahují kompresních poměrů v rozmezí 2-5 % ve srovnání s Protocol Buffers.

4. **Komprese a kódování**: Aplikujte techniky komprese a kódování ke snížení velikosti dat přenášených mezi komponenty workflow. Kompresní algoritmy jako gzip nebo Brotli mohou výrazně snížit využití síťové šířky pásma a zlepšit celkový výkon systému.

Zohledněním aspektů škálovatelnosti a výkonu během návrhu a implementace systémů inteligentní orchestrace workflow můžete zajistit, že váš systém zvládne vysoké objemy souběžných workflow, optimalizuje výkon komponent založených na umělé inteligenci a bezproblémově se přizpůsobí rostoucím požadavkům. Pro udržení výkonu a odezvy systému při zvyšující se zátěži a složitosti v průběhu času je nezbytné průběžné monitorování, profilování a optimalizace.

Testování a validace workflow

Testování a validace jsou klíčové aspekty vývoje a údržby systémů inteligentní orchestrace workflow. Vzhledem ke komplexní povaze workflow založených na umělé inteligenci je nezbytné zajistit, aby každá komponenta fungovala podle očekávání, celkové workflow se chovalo správně a rozhodnutí AI byla přesná a spolehlivá. V této části prozkoumáme různé techniky a aspekty testování a validace inteligentních workflow.

Jednotkové testování komponent workflow

Jednotkové testování zahrnuje testování jednotlivých komponent workflow izolovaně pro ověření jejich správnosti a robustnosti. Při jednotkovém testování komponent založených na AI zvažte následující:

1. **Validace vstupu**: Otestujte schopnost komponenty zpracovat různé typy vstupů, včetně platných a neplatných dat. Ověřte, že komponenta elegantně zvládá krajní případy a poskytuje odpovídající chybové zprávy nebo výjimky.

2. **Ověření výstupu**: Potvrďte, že komponenta produkuje očekávaný výstup pro danou sadu vstupů. Porovnejte skutečný výstup s očekávanými výsledky pro zajištění správnosti.

3. **Zpracování chyb**: Otestujte mechanismy zpracování chyb komponenty simulací různých chybových scénářů, jako je neplatný vstup, nedostupnost zdrojů nebo neočekávané výjimky. Ověřte, že komponenta zachytí a správně zpracuje chyby.

4. **Hraniční podmínky**: Otestujte chování komponenty při hraničních podmínkách, jako je prázdný vstup, maximální velikost vstupu nebo extrémní hodnoty. Zajistěte, že komponenta zvládá tyto podmínky elegantně bez pádu nebo produkování nesprávných výsledků.

Zde je příklad jednotkového testu pro komponentu workflow v Ruby pomocí testovacího frameworku 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
```

V tomto příkladu je komponenta `OrderValidator` testována pomocí dvou testovacích případů: jednoho pro platnou objednávku a druhého pro neplatnou objednávku. Testovací případy ověřují, že metoda `validate` vrací očekávanou booleovskou hodnotu na základě platnosti objednávky.

Integrační testování interakcí pracovního postupu

Integrační testování se zaměřuje na ověřování interakcí a toku dat mezi různými komponentami pracovního postupu. Zajišťuje, že komponenty spolupracují bezproblémově a produkují očekávané výsledky. Při integračním testování inteligentních pracovních postupů zvažte následující:

1. **Interakce komponent**: Testujte komunikaci a výměnu dat mezi komponentami pracovního postupu. Ověřte, že výstup jedné komponenty je správně předán jako vstup další komponentě v pracovním postupu.

2. **Konzistence dat**: Zajistěte, že data zůstávají konzistentní a přesná během průchodu pracovním postupem. Ověřte, že transformace dat, výpočty a agregace jsou prováděny správně.

3. **Propagace výjimek**: Testujte, jak jsou výjimky a chyby propagovány a zpracovávány napříč komponentami pracovního postupu. Ověřte, že výjimky jsou zachyceny, zaznamenány a správně zpracovány, aby nedošlo k narušení pracovního postupu.

4. **Asynchronní chování**: Pokud pracovní postup zahrnuje asynchronní komponenty nebo paralelní vykonávání, testujte mechanismy koordinace a synchronizace. Zajistěte, že pracovní postup se chová správně v souběžných a asynchronních scénářích.

Zde je příklad integračního testu pro pracovní postup v Ruby s využitím testovacího frameworku 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

end
```

V tomto příkladu je `OrderProcessingWorkflow` testován ověřením interakcí mezi různými komponenty workflow. Testovací případ nastavuje očekávání pro chování každé komponenty a zajišťuje, že workflow úspěšně zpracuje objednávku a odpovídajícím způsobem aktualizuje její stav.

Testování rozhodovacích bodů AI

Testování rozhodovacích bodů AI je klíčové pro zajištění přesnosti a spolehlivosti workflows poháněných umělou inteligencí. Při testování rozhodovacích bodů AI zvažte následující:

1. **Přesnost rozhodování**: Ověřte, že komponenta AI činí přesná rozhodnutí na základě vstupních dat a natrénovaného modelu. Porovnejte rozhodnutí AI s očekávanými výsledky nebo referenčními daty.

2. **Krajní případy**: Otestujte chování komponenty AI v krajních případech a neobvyklých scénářích. Ověřte, že komponenta AI zvládá tyto případy elegantně a činí rozumná rozhodnutí.

3. **Předpojatost a spravedlivost**: Vyhodnoťte komponentu AI z hlediska potenciální předpojatosti a zajistěte, že činí spravedlivá a nezaujatá rozhodnutí. Otestujte komponentu s různorodými vstupními daty a analyzujte výsledky na přítomnost jakýchkoliv diskriminačních vzorců.

4. **Vysvětlitelnost**: Pokud komponenta AI poskytuje vysvětlení nebo zdůvodnění svých rozhodnutí, ověřte správnost a srozumitelnost těchto vysvětlení. Zajistěte, že vysvětlení odpovídají základnímu rozhodovacímu procesu.

Zde je příklad testování rozhodovacího bodu AI v Ruby s použitím testovacího frameworku 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
```

V tomto příkladu je AI komponenta `FraudDetector` testována dvěma testovacími případy: jedním pro podvodnou transakci a druhým pro legitimní transakci. Testovací

případy ověřují, že metoda `detect_fraud` vrací očekávanou booleovskou hodnotu na základě charakteristik transakce.

End-to-End testování

End-to-end testování zahrnuje testování celého pracovního postupu od začátku do konce, simuluje reálné scénáře a uživatelské interakce. Zajišťuje, že pracovní postup se chová správně a produkuje požadované výsledky. Při provádění end-to-end testování pro inteligentní pracovní postupy zvažte následující:

1. **Uživatelské scénáře**: Identifikujte běžné uživatelské scénáře a otestujte chování pracovního postupu v těchto scénářích. Ověřte, že pracovní postup správně zpracovává uživatelské vstupy, činí vhodná rozhodnutí a produkuje očekávané výstupy.

2. **Validace dat**: Zajistěte, že pracovní postup validuje a čistí uživatelské vstupy, aby se předešlo nekonzistencím v datech nebo bezpečnostním zranitelnostem. Otestujte pracovní postup s různými typy vstupních dat, včetně platných i neplatných dat.

3. **Zotavení z chyb**: Otestujte schopnost pracovního postupu zotavit se z chyb a výjimek. Simulujte chybové scénáře a ověřte, že pracovní postup je zvládá elegantně, zaznamenává chyby a provádí příslušné kroky k zotavení.

4. **Výkon a škálovatelnost**: Vyhodnoťte výkon a škálovatelnost pracovního postupu při různých podmínkách zatížení. Otestujte pracovní postup s velkým objemem souběžných požadavků a změřte doby odezvy, využití zdrojů a celkovou stabilitu systému.

Zde je příklad end-to-end testu pro pracovní postup v Ruby s využitím testovacího frameworku RSpec a knihovny Capybara pro simulaci uživatelských interakcí:

```
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
```

V tomto příkladu end-to-end test simuluje uživatele, který zadává objednávku přes webové rozhraní. Vyplňuje požadovaná pole formuláře, odesílá objednávku a ověřuje, že objednávka je úspěšně zpracována, zobrazuje příslušnou potvrzující zprávu a aktualizuje stav objednávky v databázi.

Průběžná integrace a nasazení

Pro zajištění spolehlivosti a udržovatelnosti inteligentních workflow se doporučuje integrovat testování a validaci do pipeline průběžné integrace a nasazení (CI/CD). To umožňuje automatizované testování a validaci změn workflow před jejich nasazením do produkce. Zvažte následující postupy:

1. **Automatizované spouštění testů**: Nakonfigurujte CI/CD pipeline tak, aby automaticky spouštěla sadu testů při každé změně v kódové základně workflow. Tím zajistíte, že případné regrese nebo selhání budou odhaleny již v počátečních fázích vývoje.

2. **Sledování testovacího pokrytí**: Měřte a sledujte testovací pokrytí komponent workflow a bodů AI rozhodování. Usilujte o vysoké testovací pokrytí, abyste zajistili důkladné otestování kritických cest a scénářů.

3. **Průběžná zpětná vazba**: Integrujte výsledky testů a metriky kvality kódu do vývojového workflow. Poskytujte vývojářům průběžnou zpětnou vazbu o stavu testů, kvalitě kódu a jakýchkoli problémech zjištěných během CI/CD procesu.

4. **Staging prostředí**: Nasaďte workflow do staging prostředí, která věrně kopírují produkční prostředí. Proveďte dodatečné testování a validaci ve staging prostředí, abyste odhalili případné problémy související s infrastrukturou, konfigurací nebo integrací dat.

5. **Mechanismy rollbacku**: Implementujte mechanismy rollbacku pro případ selhání nasazení nebo zjištění kritických problémů v produkci. Zajistěte, aby workflow mohlo být rychle vráceno na předchozí stabilní verzi, čímž se minimalizuje výpadek a dopad na uživatele.

Začleněním testování a validace do celého vývojového cyklu inteligentních workflow mohou organizace zajistit spolehlivost, přesnost a udržovatelnost svých systémů založených na AI. Pravidelné testování a validace pomáhají odhalit chyby, předcházet regresím a budovat důvěru v chování a výsledky workflow.

Část 2: Vzory

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Prompt Engineering

Řetězení myšlenek

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Jak to funguje

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Příklady

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Generování obsahu

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Vytváření strukturovaných entit

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Vedení LLM agenta

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Výhody a aspekty k zvážení

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Přepínač režimů

Jak to funguje

Kdy to použít

Příklad

Přiřazení role

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Jak to funguje

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Kdy to použít

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Příklady

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Prompt Object

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Jak to funguje

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Šablona promptu

Jak to funguje

Výhody a úvahy

Kdy ji použít:

Příklad

Structured IO

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Jak to funguje

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Škálování Structured IO

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Výhody a úvahy

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Řetězení promptů

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Jak to funguje

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Kdy to použít

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Příklad: Onboarding v Olympii

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Přepisovač promptů

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Jak to funguje

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Příklad

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Ohraničení odpovědi

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Jak to funguje

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Výhody a úvahy

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Ošetření chyb

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Analyzátor dotazů

Jak to funguje

Implementace

Označování slovních druhů (POS) a rozpoznávání pojmenovaných entit (NER)

Klasifikace záměru

Extrakce klíčových slov

Výhody

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Přepisovač dotazů

Jak to funguje

Příklad

Výhody

Ventriloquist

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Jak to funguje

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Kdy to použít

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Příklad

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Diskrétní komponenty

Predicate

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Jak to funguje

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Kdy jej použít

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Příklad

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

API Fasáda

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Jak to funguje

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Klíčové výhody

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Kdy ji použít

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Příklad

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Autentizace a autorizace

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Zpracování požadavků

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Formátování odpovědí

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Zpracování chyb a krajních případů

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Úvahy o škálovatelnosti a výkonu

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Porovnání s jinými návrhovými vzory

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Interpret výsledků

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Jak to funguje

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Kdy jej použít

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Příklad

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Virtuální stroj

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Jak to funguje

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Kdy jej použít

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Příklad

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Za oponou magie

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Specifikace a testování

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Specifikace chování

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Psaní testovacích případů

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Příklad: Testování komponenty překladače

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Přehrávání HTTP interakcí

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Human In The Loop (HITL)

Vysokoúrovňové vzory

Hybridní inteligence

Adaptivní odezva

Přepínání rolí mezi člověkem a UI

Eskalace

Jak to funguje

Klíčové výhody

Praktická aplikace: Zdravotnictví

Zpětnovazební smyčka

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Jak to funguje

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Aplikace a příklady

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Pokročilé techniky integrace lidské zpětné vazby

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Pasivní radiace informací

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Jak to funguje

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Kontextové zobrazení informací

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Proaktivní upozornění

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Vysvětlující poznatky

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Interaktivní průzkum

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Klíčové výhody

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Aplikace a příklady

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Kolaborativní rozhodování (CDM)

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Jak to funguje

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Příklad

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Kontinuální učení

Jak to funguje

Aplikace a příklady

Příklad

Etické aspekty

Role HITL při zmírňování rizik AI

Technologický pokrok a výhled do budoucnosti

Výzvy a omezení systémů HITL

Inteligentní zpracování chyb

Tradiční přístupy ke zpracování chyb

Kontextuální diagnostika chyb

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Jak to funguje

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Promptové inženýrství pro kontextuální diagnostiku chyb

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Generování rozšířené o vyhledávání pro kontextovou diagnostiku chyb

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Inteligentní hlášení chyb

Prediktivní prevence chyb

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Jak to funguje

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Chytré zotavení z chyb

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Jak to funguje

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Personalizovaná komunikace chyb

Jak to funguje

Adaptivní workflow zpracování chyb

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Jak to funguje

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Kontrola kvality

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Eval

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Problém

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Řešení

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Jak to funguje

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Příklad

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Důležité aspekty

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Porozumění zlatým referencím

Jak fungují evaluace bez referencí

Ochranný mechanismus

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Problém

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Řešení

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Jak to funguje

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Příklad

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Důležité aspekty

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Ochranné mechanismy a vyhodnocení: Dvě strany téže mince

Zaměnitelnost ochranných mechanismů a evaluací bez reference

Implementace duálních ochranných mechanismů a evaluací

Glosář

Glosář

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

W

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Z

Tento obsah není k dispozici v ukázkové knize. Kniha lze zakoupit na Leanpub na http://leanpub.com/patterns-of-application-development-using-ai-cs.

Index