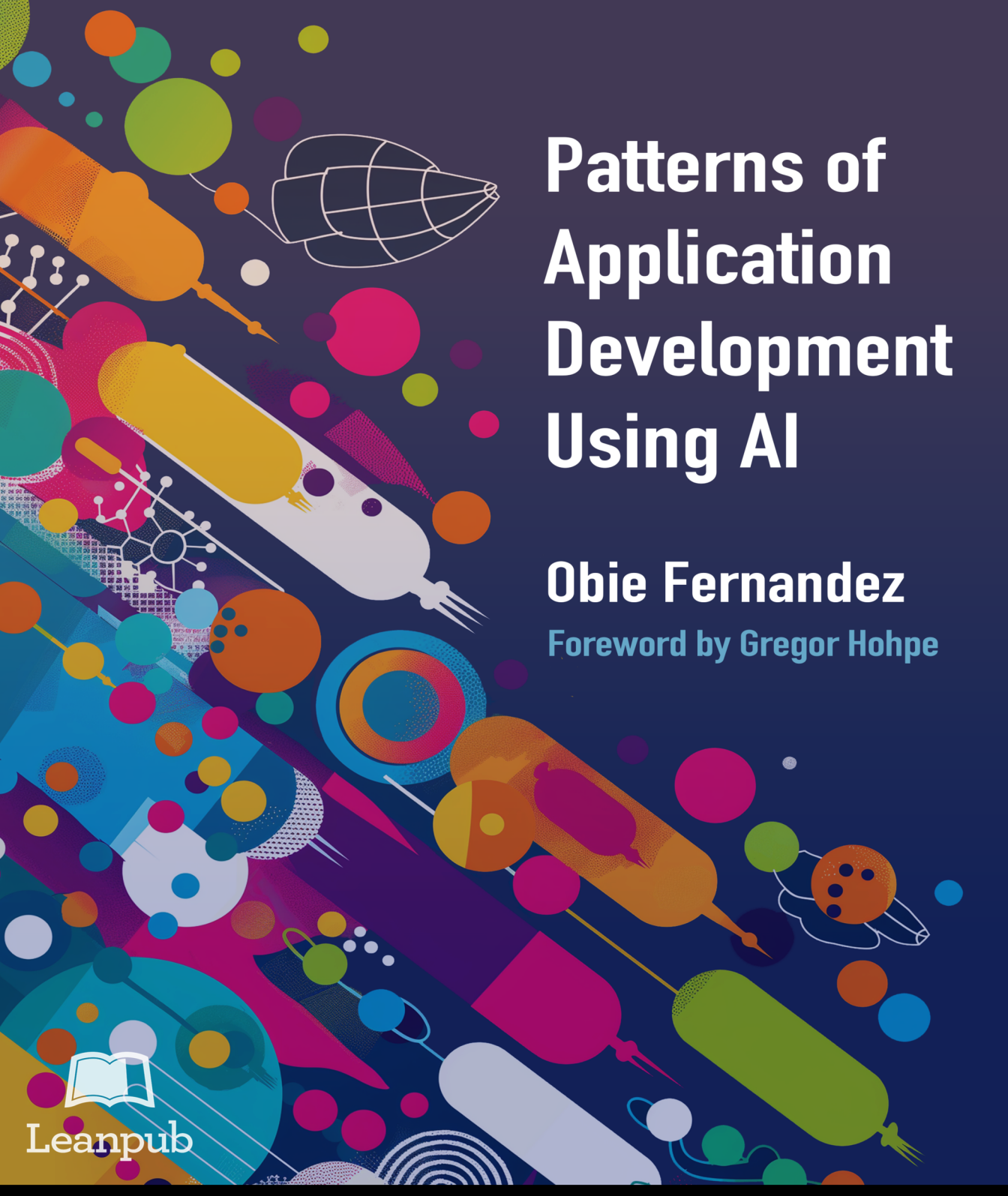


Magyar Kiadás

AI Segítségével Történő Alkalmazásfejlesztési Minták (Magyar Kiadás)

Obie Fernandez

Ez a könyv megvásárolható:
http://leanpub.com/patterns-of-application-development-using-ai-hu

Jelen kiadás megjelenésének időpontja: 2025-01-23

Ez egy Leanpub könyv. A Leanpub a szerzőket és kiadókat egy rugalmas kiadási folyamattal segíti munkájukban. A rugalmas kiadás lényege, hogy az előkészületben levő elektronikus könyveket könnyen kezelhető, egyszerű eszközökkel dolgozzuk ki, több iteráción keresztül, az olvasók folyamatos visszajelzései alapján egészen addig, amíg el jutunk a tartalmilag megfelelő és kellően nagy érdeklődést magához vonzó változathoz.

Twitteld ki a könyvet!

Kérjük segítsd a szerzőt (Obie Fernandez) azzal, hogy megosztod a könyvét Twitteren!

A javasolt hashtag: #poaduai.

Derítsd ki, mit mondtak mások erről a könyvről azzal, hogy rákattintasz a linkre és ezzel rákeresel erre a hashtagre Twitteren:

#poaduai

Vagány királynőmnek, múzsámnak, fényemnek és szerelmemnek, Victoriának

A szerző további könyvei: 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

Tartalomjegyzék

Előszó Gregor Hohpe tollából

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Előszó

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

A könyvről

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

A kódpéldákról

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Amit nem tárgyalok

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Kinek szól ez a könyv

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Közös Szókincs Kialakítása

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Kapcsolódás

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Köszönetnyilvánítás

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Mi a helyzet az illusztrációkkal?

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

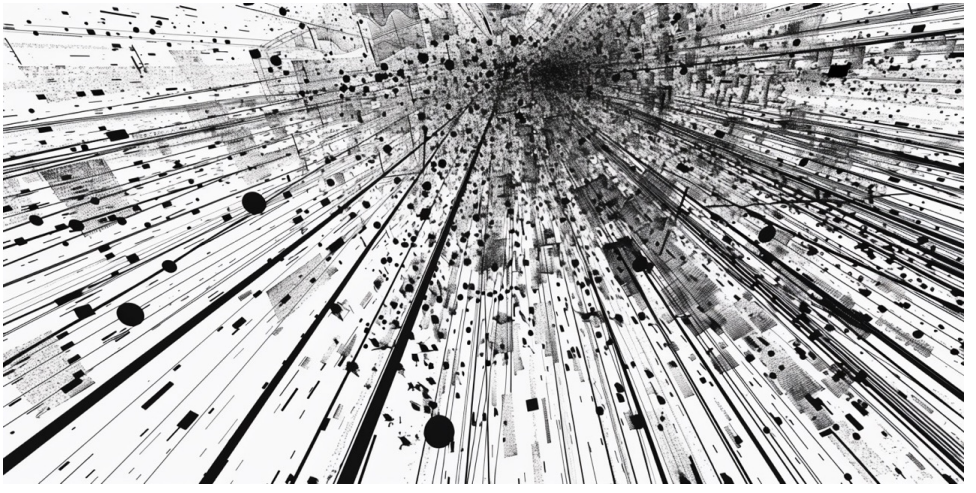
A Lean Publishing-ról

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

A szerzőről

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Bevezetés



Ha alig várja, hogy elkezdje integrálni az AI Nagy Nyelvi Modelleket (LLM) a programozási projektjeibe, nyugodtan ugorjon egyenesen a későbbi fejezetekben bemutatott mintákra és kódpéldákra. Azonban hogy teljes mértékben értékelni tudja ezen minták erejét és potenciálját, érdemes egy pillanatra megállni és megérteni a tágabb kontextust és az általuk képviselt egységes megközelítést.

A minták nem csupán elszigetelt technikák gyűjteménye, hanem egy egységes keretrendszer az AI alkalmazásainkba történő integrálásához. Én a Ruby on Rails keretrendszert használom, de ezek a minták gyakorlatilag bármely más programozási környezetben működnek. Számos szempontot lefednek az adatkezeléstől és teljesítményoptimalizálástól kezdve a felhasználói élményig és biztonságig, átfogó eszköztárat biztosítva a hagyományos programozási gyakorlatok AI képességekkel történő fejlesztéséhez.

A minták minden kategóriája egy-egy specifikus kihívást vagy lehetőséget céloz meg, amely az AI komponensek alkalmazásba történő beépítésekor merül fel. A minták közötti kapcsolatok és szinergiák megértésével megalapozott döntéseket hozhat arról, hogy hol és hogyan alkalmazza leghatékonyabban az AI-t.

A minták soha nem előíró jellegű megoldások, és nem is szabad őket akként kezelni. Adaptálható építőelemeknek szánják őket, amelyeket az Ön egyedi alkalmazásának egyedi követelményeihez és korlátaihoz kell igazítani. Ezen minták sikeres alkalmazása (mint bármely más minta a szoftverek területén) a probléma területének, a felhasználói igényeknek és a projekt átfogó technikai architektúrájának mély megértésén alapul.

Gondolatok a szoftverarchitektúráról

Az 1980-as években kezdtem programozni és részt vettem a hacker közösségben, és soha nem veszítettem el a hacker szemléletemet, még azután sem, hogy professzionális szoftverfejlesztővé váltam. A kezdetektől fogva egészséges szkepticizmussal viseltettem azzal kapcsolatban, hogy az elefántcsonttornyukban ülő szoftverarchitektusok valójában milyen értéket teremtenek.

Az egyik ok, amiért személyesen annyira lelkes vagyok az AI technológia ezen új, erőteljes hullámának változásaival kapcsolatban, az a *szoftverarchitektúrára* gyakorolt hatása. Megkérdőjelezi a hagyományos elképzeléseket arról, hogy mi számít "helyes" módszernek szoftverprojektjeink tervezésében és megvalósításában. Azt is megkérdőjelezi, hogy az architektúrát továbbra is elsősorban úgy lehet-e tekinteni, mint *a rendszer nehezen változtatható részeit*, mivel az AI-fejlesztés minden eddiginél könnyebbé teszi a projekt bármely részének módosítását, bármikor.

Talán a szoftverfejlesztés "posztmodern" megközelítésének csúcséveit éljük. Ebben a kontextusban a posztmodern egy alapvető elmozdulást jelent a hagyományos paradigmáktól, ahol a fejlesztők voltak felelősek minden egyes kódsor megírásáért és karbantartásáért. Ehelyett elfogadja azt az elképzelést, hogy feladatokat delegálunk,

például adatmanipulációt, komplex algoritmusokat és akár teljes alkalmazáslogikai részeket is külső könyvtáraknak és API-knak. Ez a posztmodern váltás jelentős eltérést jelent az alkalmazások alapoktól történő felépítésének hagyományos bölcsességétől, és arra készteti a fejlesztőket, hogy újragondolják szerepüket a fejlesztési folyamatban.

Mindig is azt vallottam, hogy a jó programozók csak azt a kódot írják meg, amit feltétlenül szükséges, Larry Wall és más hozzá hasonló hacker nagyságok tanításai alapján. A megírt kód mennyiségének minimalizálásával gyorsabban haladhatunk, csökkenthetjük a hibák előfordulási felületét, egyszerűsíthetjük a karbantartást és javíthatjuk az alkalmazások általános megbízhatóságát. A kevesebb kód lehetővé teszi, hogy az alapvető üzleti logikára és felhasználói élményre koncentráljunk, miközben más feladatokat más szolgáltatásokra bízunk.

Most, hogy az AI-vezérelt rendszerek olyan feladatokat is képesek kezelni, amelyek korábban kizárólag az ember által írt kód területéhez tartoztak, még produktívabbnak és rugalmasabbnak kellene lennünk, minden eddiginél nagyobb hangsúlyt fektetve az üzleti érték és a felhasználói élmény megteremtésére.

Természetesen vannak kompromisszumok a projekt nagy részeinek AI rendszerekre való delegálásában, mint például a potenciális kontrollvesztés és a robusztus monitoring és visszajelzési mechanizmusok szükségessége. Ezért új készségekre és tudásra van szükség, beleértve legalább az AI működésének alapvető megértését.

Mi az a Nagy Nyelvi Modell?

A Nagy Nyelvi Modellek (LLM-ek) olyan típusú mesterséges intelligencia modellek, amelyek jelentős figyelmet kaptak az elmúlt években, különösen az OpenAI által 2020-ban kiadott GPT-3 óta. Az LLM-eket arra tervezték, hogy figyelemre méltó pontossággal és folyékonysággal dolgozzák fel, értsék meg és generálják az emberi nyelvet. Ebben a részben röviden áttekintjük, hogyan működnek az LLM-ek és miért alkalmasak intelligens rendszerkomponensek építésére.

Alapjában véve az LLM-ek mély tanulási algoritmusokon, konkrétan neurális hálózatokon alapulnak. Ezek a hálózatok összekapcsolt csomópontokból vagy neuronokból állnak, amelyek információt dolgoznak fel és továbbítanak. Az LLM-ek számára a választott architektúra gyakran a Transformer modell, amely rendkívül hatékonynak bizonyult a szöveghez hasonló szekvenciális adatok kezelésében.

A transformer modellek a figyelmi mechanizmuson alapulnak, és elsősorban szekvenciális adatokkal kapcsolatos feladatokra használják őket, mint például a természetes nyelvfeldolgozás. A transformerek a bemeneti adatokat egyszerre dolgozzák fel, nem pedig szekvenciálisan, ami lehetővé teszi számukra a hosszú távú függőségek hatékonyabb megragadását. Olyan figyelmi mechanizmus rétegekkel rendelkeznek, amelyek segítik a modellt abban, hogy a bemeneti adatok különböző részeire összpontosítson a kontextus és a kapcsolatok megértéséhez.

A nagy nyelvi modellek képzési folyamata során a modellt hatalmas mennyiségű szöveges adatnak teszik ki, például könyveknek, cikkeknek, weboldalaknak és kódtáraknak. A képzés során a modell megtanulja felismerni a szövegben található mintákat, kapcsolatokat és struktúrákat. Megragadja a nyelv statisztikai tulajdonságait, például a nyelvtani szabályokat, a szókapcsolatokat és a kontextuális jelentéseket.

A nagy nyelvi modellek képzésében használt egyik kulcsfontosságú technika a felügyelet nélküli tanulás. Ez azt jelenti, hogy a modell explicit címkézés vagy irányítás nélkül tanul az adatokból. Önállóan fedezi fel a mintákat és reprezentációkat a tanító adatokban található szavak és kifejezések együttes előfordulásának elemzésével. Ez lehetővé teszi a nagy nyelvi modellek számára, hogy mély megértést alakítsanak ki a nyelvről és annak összetettségéről.

A nagy nyelvi modellek egy másik fontos aspektusa a *kontextus* kezelésének képessége. Egy szövegrész feldolgozásakor a nagy nyelvi modellek nem csak az egyes szavakat veszik figyelembe, hanem a környező kontextust is. Figyelembe veszik az előző szavakat, mondatokat és akár bekezdéseket is a szöveg jelentésének és szándékának megértéséhez. Ez a kontextuális megértés teszi lehetővé a nagy nyelvi modellek számára, hogy

koherens és releváns válaszokat generáljanak. Az egyik fő módja annak, hogy egy adott nagy nyelvi modell képességeit értékeljük, az, hogy figyelembe vesszük, mekkora kontextust tudnak figyelembe venni a válaszok generálásához.

A képzés után a nagy nyelvi modellek számos nyelvi feladatra használhatók. Képesek emberszerű szöveget generálni, kérdéseket megválaszolni, dokumentumokat összefoglalni, nyelveket fordítani, sőt kódot írni. A nagy nyelvi modellek sokoldalúsága értékessé teszi őket olyan intelligens rendszerkomponensek építésében, amelyek képesek a felhasználókkal kommunikálni, szöveges adatokat feldolgozni és elemezni, valamint értelmes kimeneteket generálni.

A nagy nyelvi modellek alkalmazásarchitektúrába történő beépítésével olyan AI-komponenseket hozhat létre, amelyek megértik és feldolgozzák a felhasználói bevitelt, dinamikus tartalmat generálnak, és intelligens ajánlásokat vagy műveleteket biztosítanak. De a nagy nyelvi modellekkel való munka során gondosan mérlegelni kell az erőforrásigényeket és a teljesítménybeli kompromisszumokat. A nagy nyelvi modellek számításigényesek, és jelentős feldolgozási teljesítményt és memóriát (más szóval pénzt) igényelhetnek a működéshez. A legtöbbünknek értékelnie kell a nagy nyelvi modellek alkalmazásainkba való integrálásának költségvonzatait, és ennek megfelelően kell cselekednie.

A következtetés megértése

A következtetés arra a folyamatra utal, amelynek során egy modell új, korábban nem látott adatok alapján előrejelzéseket vagy kimeneteket generál. Ez az a fázis, amikor a betanított modellt felhasználói bemenetek alapján döntések meghozatalára vagy szöveg, képek vagy egyéb tartalom generálására használják.

A tanítási fázis során egy AI modell nagy adathalmazból tanul úgy, hogy paramétereit az előrejelzéseiben lévő hiba minimalizálása érdekében állítja be. A betanítás után a modell képes alkalmazni a tanultakat új adatokra. A következtetés az a folyamat, ahogyan a modell a megtanult mintákat és tudást felhasználja kimenetek generálására.

A nagy nyelvi modellek esetében a következtetés magában foglalja egy prompt vagy bemeneti szöveg feldolgozását és egy koherens, kontextuálisan releváns válasz előállítását *tokenek* folyamaként (amelyekről hamarosan beszélünk). Ez lehet egy kérdés megválaszolása, egy mondat befejezése, egy történet generálása vagy szöveg fordítása, sok más feladat mellett.

Ellentétben azzal, ahogyan Ön és én gondolkodunk, egy AI modell "gondolkodása" a következtetés során egyetlen állapotmentes műveletben történik. Azaz a gondolkodása a generálási folyamatra korlátozódik. Szó szerint hangosan kell gondolkodnia, mintha feltettem volna Önnek egy kérdést, és csak "tudatfolyam" stílusban fogadnék el választ.

A nagy nyelvi modellek sokféle méretben és változatban léteznek

Bár gyakorlatilag minden népszerű nagy nyelvi modell ugyanazon az alap transformer architektúrán alapul és hatalmas szövegadathalmazokon van betanítva, különböző méretekben érhetők el és különböző célokra vannak finomhangolva. Egy nagy nyelvi modell mérete, amit a neurális hálózatában lévő paraméterek száma határoz meg, nagy hatással van a képességeire. A több paraméterrel rendelkező nagyobb modellek, mint például a GPT-4, amelyről azt pletykálják, hogy 1-2 billió paraméterrel rendelkezik, általában tudásanyagban gazdagabbak és képességeikben fejlettebbek, mint a kisebb modellek. Azonban a nagyobb modellek sokkal több számítási teljesítményt igényelnek a futtatáshoz, ami magasabb költségeket jelent, amikor API-hívásokon keresztül használja őket.

A nagy nyelvi modellek gyakorlatiasabbá és specifikus felhasználási esetekre szabottá tétele érdekében az alapmodelleket gyakran célzottabb adathalmazokon finomhangolják. Például egy nagy nyelvi modellt párbeszédek nagy gyűjteményén taníthatnak be, hogy specializálják beszélgető AI-ra. Másokat kódon tanítanak be, hogy

programozási tudással ruházzák fel őket. Vannak még olyan modellek is, amelyeket kifejezetten szerepjáték-stílusú felhasználói interakciókra tanítottak be!

Visszakeresés vs Generatív Modellek

A nagy nyelvi modellek (NNM-ek) világában két fő megközelítés létezik a válaszok generálására: a visszakeresésen alapuló modellek és a generatív modellek. Mindkét megközelítésnek megvannak a maga erősségei és gyengeségei, és a köztük lévő különbségek megértése segíthet a megfelelő modell kiválasztásában az adott felhasználási esethez.

Visszakeresésen Alapuló Modellek

A visszakeresésen alapuló modellek, más néven információ-visszakereső modellek, úgy generálnak válaszokat, hogy egy nagy, előre létező szövegadatbázisban keresnek, és a bemeneti lekérdezés alapján kiválasztják a legrelevánsabb részleteket. Ezek a modellek nem generálnak új szöveget a semmiből, hanem inkább az adatbázisból származó részleteket fűzik össze egy koherens válasszá.

A visszakeresésen alapuló modellek egyik fő előnye, hogy tényszerűen pontos és naprakész információkat tudnak szolgáltatni. Mivel egy válogatott szövegadatbázisra támaszkodnak, megbízható forrásokból tudnak releváns információkat kinyerni és bemutatni a felhasználónak. Ez különösen alkalmassá teszi őket olyan alkalmazásokhoz, amelyek pontos, tényszerű válaszokat igényelnek, mint például a kérdésmegválaszoló rendszerek vagy tudásbázisok.

Azonban a visszakeresésen alapuló modelleknek vannak korlátaik. Csak olyan jók lehetnek, mint az az adatbázis, amelyben keresnek, így az adatbázis minősége és lefedettsége közvetlenül befolyásolja a modell teljesítményét. Emellett ezek a modellek nehezen tudnak koherens és természetesen hangzó válaszokat generálni, mivel az adatbázisban elérhető szövegekre korlátozódnak.

Ebben a könyvben nem tárgyaljuk a tisztán visszakeresésen alapuló modellek használatát.

Generatív Modellek

A generatív modellek ezzel szemben a tanulás során elsajátított mintázatok és összefüggések alapján a semmiből hoznak létre új szöveget. Ezek a modellek a nyelv megértését használják fel arra, hogy a bemeneti prompthoz szabott új válaszokat generáljanak.

A generatív modellek fő erőssége, hogy kreatív, koherens és kontextuálisan releváns szöveget tudnak létrehozni. Képesek nyitott beszélgetéseket folytatni, történeteket generálni, sőt kódot írni. Ez ideálissá teszi őket olyan alkalmazásokhoz, amelyek nyitottabb és dinamikusabb interakciókat igényelnek, mint például a chatbotok, tartalomkészítés és kreatív írást segítő asszisztensek.

Azonban a generatív modellek néha következetlen vagy tényszerűen helytelen információkat produkálhatnak, mivel inkább a tanulás során elsajátított mintázatokra támaszkodnak, nem pedig egy válogatott tényadatbázisra. Hajlamosabbak lehetnek az elfogultságra és a hallucinációkra is, olyan szöveget generálva, ami hihetőnek tűnik, de nem feltétlenül igaz.

A generatív NNM-ek példái közé tartozik az OpenAI GPT sorozata (GPT-3, GPT-4) és az Anthropic Claude-ja.

Hibrid Modellek

Számos kereskedelmi forgalomban kapható NNM ötvözi mind a visszakeresési, mind a generatív megközelítéseket egy hibrid modellben. Ezek a modellek visszakeresési technikákat használnak a releváns információk megtalálására az adatbázisból, majd generatív technikákat alkalmaznak ezen információk koherens válasszá történő szintetizálására.

A hibrid modellek célja, hogy ötvözzék a visszakeresésen alapuló modellek tényszerű pontosságát a generatív modellek természetes nyelvgenerálási képességeivel. Megbízhatóbb és naprakészebb információkat tudnak szolgáltatni, miközben megőrzik a nyitott beszélgetések folytatásának képességét.

A visszakeresésen alapuló és a generatív modellek közötti választáskor figyelembe kell venni az alkalmazás specifikus követelményeit. Ha az elsődleges cél pontos, tényszerű információk szolgáltatása, akkor a visszakeresésen alapuló modell lehet a legjobb választás. Ha az alkalmazás nyitottabb és kreatívabb interakciókat igényel, akkor egy generatív modell lehet megfelelőbb. A hibrid modellek egyensúlyt kínálnak a két megközelítés között, és jó választást jelenthetnek olyan alkalmazásokhoz, amelyek mind tényszerű pontosságot, mind természetes nyelvgenerálást igényelnek.

Végső soron a visszakeresésen alapuló és generatív modellek közötti választás az adott felhasználási esettől és a pontosság, kreativitás és rugalmasság közötti kompromisszumoktól függ. Az egyes megközelítések erősségeinek és korlátainak megértésével megalapozott döntéseket hozhatunk.

Utasításhangolás

Az utasításhangolás a beszélgetési tréning egy olyan részhalmazát képezi, amely emberek által írt promptokat és válaszokat használ arra, hogy az NNM-ek jobban értsék és kövessék az utasításokat. Íme néhány egyszerű példa annak szemléltetésére, hogy az emberek által írt promptokon és válaszokon alapuló utasításhangolás hogyan teszi az NNM-eket jobbá az utasítások megértésében és követésében:

1. Példa: Ügyfélszolgálat

Utasítás: "Adj választ egy visszatérítést kérő ügyfélnek."

Ember által írt prompt: "Szeretnék visszatérítést kérni a legutóbbi vásárlásomra. A termék nem felelt meg az elvárásaimnak."

Ember által írt válasz: "Sajnálattal hallom, hogy a termék nem felelt meg az elvárásainak. A visszatérítés feldolgozásához kérem, adja meg a rendelési számát és a visszaküldés okát. Átnézzük a kérését és 2-3 munkanapon belül visszajelzünk."

2. Példa: Oktatási Segítség

Utasítás: "Magyarázd el a fotoszintézis fogalmát egy 10 éves gyereknek."

Emberi kérdés: "El tudnád magyarázni, hogyan készítik a növények a táplálékukat?"

Emberi válasz: "Persze! A növények a fotoszintézis nevű folyamat során állítják elő a táplálékukat. Napfényt, vizet és szén-dioxidot használnak fel, hogy táplálékot és oxigént hozzanak létre. A növény levelében van egy különleges rész, a klorofill, ami segít nekik befogni a napfényt."

3. példa: Személyi asszisztens

Utasítás: "Ütemezz be egy megbeszélést jövő keddre délután 3 órára."

> Emberi kérdés: “Be tudnál ütemezni egy megbeszélést keddre délután 3 órára?”
>
> Emberi válasz: “Persze, beütemeztem neked egy megbeszélést jövő kedd délután 3 órára. Van még valami, amiben segíthetek?”

Az eredmény a különböző méretű és különböző szakterületekre specializálódott LLM-ek változatos ökoszisztémája. Az 1-7 milliárd paraméteres kisebb modellek jó általános nyelvi képességeket nyújtanak, miközben hatékonyabban futtathatók.

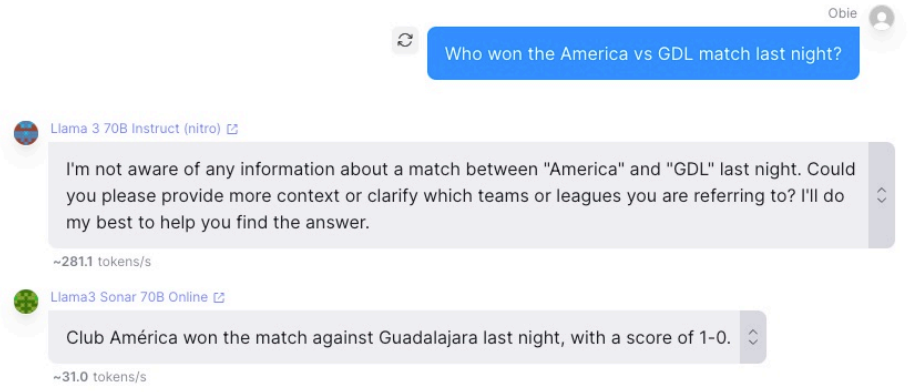
- Mistral 7B
- Llama 3 8B
- Gemma 7B

A körülbelül 30-70 milliárd paraméteres közepes méretű modellek erősebb következtetési és utasításkövetési képességeket kínálnak.

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

Amikor egy alkalmazásba LLM-et kívánunk beépíteni, egyensúlyt kell teremtenünk a modell képességei és olyan gyakorlati tényezők között, mint a költség, késleltetés, kontextushossz és tartalomszűrés. Az egyszerűbb nyelvi feladatokhoz gyakran a kisebb, utasításra hangolt modellek jelentik a legjobb választást, míg a legnagyobb modellekre összetett következtetésekhez vagy elemzésekhez lehet szükség. A modell tanítási adatai szintén fontos szempontot jelentenek, mivel ezek határozzák meg a modell tudásának záró dátumát.

Bizonyos modellek, mint például néhány a Perplexity kínálatából, valós idejű információforrásokhoz kapcsolódnak, így gyakorlatilag nincs tudás záró dátumuk. Amikor kérdéseket teszünk fel nekik, önállóan dönthetnek úgy, hogy webes kereséseket végeznek és tetszőleges weboldalakat töltenek be a válasz generálásához.



ábra 1. Llama3 online hozzáféréssel és anélkül

Végeredményben nincs univerzális LLM. A modellméret, architektúra és tanítás változatainak megértése kulcsfontosságú a megfelelő modell kiválasztásához egy adott felhasználási esethez. A különböző modellek kipróbálása az egyetlen gyakorlati módja annak, hogy kiderítsük, melyek nyújtják a legjobb teljesítményt az adott feladathoz.

Tokenizálás: A szöveg részekre bontása

Mielőtt egy nagy nyelvi modell feldolgozhatná a szöveget, azt kisebb egységekre, úgynevezett *tokenekre* kell bontani. A tokenek lehetnek különálló szavak, szórészek, vagy akár egyetlen karakterek is. A szöveg tokenekre bontásának folyamatát tokenizálásnak nevezzük, és ez kulcsfontosságú lépés az adatok nyelvi modell számára történő előkészítésében.

ábra 2. Ez a mondat 27 tokent tartalmaz

A különböző LLM-ek különböző tokenizálási stratégiákat használnak, amelyek jelentős hatással lehetnek a modell teljesítményére és képességeire. Néhány gyakori tokenizáló, amit az LLM-ek használnak:

- **GPT (Bájt-pár kódolás)**: A GPT tokenizálók egy bájt-pár kódolásnak (BPE) nevezett technikát használnak a szöveg alszó egységekre bontásához. A BPE iteratívan egyesíti a szövegkorpuszban leggyakrabban előforduló bájtpárokat, létrehozva egy alszó tokenekből álló szókészletet. Ez lehetővé teszi a tokenizáló számára, hogy a ritka és új szavakat gyakoribb alszó részekre bontsa. A GPT tokenizálókat olyan modellek használják, mint a GPT-3 és a GPT-4.

- **Llama (SentencePiece)**: A Llama tokenizálók a SentencePiece könyvtárat használják, amely egy felügyelet nélküli szöveg tokenizáló és detokenizáló. A SentencePiece a bemeneti szöveget Unicode karakterek sorozataként kezeli, és egy tanító korpusz alapján tanulja meg a részszavas szókészletet. Bármilyen Unicode-ban kódolható nyelvet képes kezelni, így kiválóan alkalmas többnyelvű modellekhez. A Llama tokenizálókat olyan modellek használják, mint a Meta Llama és az Alpaca.

- **SentencePiece (Unigram)**: A SentencePiece tokenizálók egy másik algoritmust is használhatnak, az úgynevezett Unigramot, amely egy részszó-regularizációs technikán alapul. Az Unigram tokenizálás az optimális részszavas szókészletet egy unigram nyelvi modell alapján határozza meg, amely valószínűségeket rendel az egyes részszó egységekhez. Ez a megközelítés szemantikailag értelmesebb részszavakat eredményezhet a BPE-hez képest. A SentencePiece Unigramot olyan modellek használják, mint a Google T5 és a BERT.

- **Google Gemini (Multimodális Tokenizálás)**: A Google Gemini olyan tokenizálási sémát használ, amely különböző adattípusok kezelésére készült, beleértve a szöveget, képeket, hangot, videókat és kódot. Ez a multimodális képesség lehetővé teszi a Gemini számára, hogy különböző információs formákat dolgozzon fel és integráljon. Figyelemre méltó, hogy a Google Gemini 1.5 Pro kontextusablaka milliónyi token kezelésére képes, ami jóval nagyobb, mint a korábbi modelleké. Ez a kiterjedt kontextusablak lehetővé teszi a modell számára, hogy nagyobb kontextust dolgozzon fel, potenciálisan pontosabb válaszokat eredményezve. Fontos azonban megjegyezni, hogy a Gemini tokenizálási sémája sokkal közelebb áll az egy token per karakter megközelítéshez, mint más modelleké. Ez azt jelenti, hogy a Gemini modellek tényleges használati költsége jelentősen magasabb lehet a vártnál, ha a GPT-hez hasonló modellekhez van szokva valaki, mivel a Google árazása karaktereken, nem pedig tokeneken alapul.

A tokenizáló választása a nyelvi modell több aspektusát is befolyásolja, többek között:

- **Szókészlet mérete**: A tokenizáló határozza meg a modell szókészletének méretét, amely az általa felismert egyedi tokenek halmaza. Egy nagyobb, finomabb szemcsézettségű szókészlet segíthet a modellnek szélesebb körű szavak és kifejezések kezelésében, és akár multimodálissá is válhat (képes több mint csak szöveg megértésére és generálására), de ez növeli a modell memóriaigényét és számítási komplexitását.
- **Ritka és ismeretlen szavak kezelése**: A részszó egységeket használó tokenizálók, mint a BPE és a SentencePiece, képesek a ritka és ismeretlen szavakat gyakoribb részszavakra bontani. Ez lehetővé teszi a modell számára, hogy megalapozott feltételezéseket tegyen olyan szavak jelentéséről, amelyekkel korábban nem találkozott, a bennük található részszavak alapján.
- **Többnyelvű támogatás**: Az olyan tokenizálók, mint a SentencePiece, amelyek bármilyen Unicode-kódolható nyelvet képesek kezelni, kiválóan alkalmasak többnyelvű modellekhez, amelyeknek több nyelven kell szöveget feldolgozniuk.

Amikor egy nyelvi modellt választunk egy adott alkalmazáshoz, fontos figyelembe venni az általa használt tokenizálót, és hogy az mennyire illeszkedik az adott feladat specifikus nyelvi feldolgozási igényeihez. A tokenizáló jelentős hatással lehet a modell képességeire a szakterület-specifikus terminológia, ritka szavak és többnyelvű szövegek kezelésében.

Kontextusméret: Mennyi Információt Tud Egy Nyelvi Modell Felhasználni Következtetés Során?

A nyelvi modellek tárgyalásakor a kontextusméret arra a szövegmennyiségre utal, amelyet egy modell figyelembe tud venni válaszai feldolgozása vagy generálása során. Lényegében azt méri, hogy mennyi információt tud a modell "megjegyezni" és felhasználni a kimenetei létrehozásához (tokenekben kifejezve). Egy nyelvi modell kontextusmérete jelentős hatással lehet a képességeire és az általa hatékonyan végrehajtható feladatok típusaira.

Mi a Kontextusméret?

Technikai szempontból a kontextusméret azt határozza meg, hogy hány tokent (szót vagy szórészletet) tud egy nyelvi modell egyetlen bemeneti szekvenciában feldolgozni. Ezt gyakran a modell "figyelmi terjedelmének" vagy "kontextusablakának" nevezik. Minél nagyobb a kontextusméret, annál több szöveget tud a modell egyszerre figyelembe venni egy válasz generálásakor vagy egy feladat végrehajtásakor.

A különböző nyelvi modellek kontextusmérete változó, néhány száz tokentől akár több millió tokenig terjedhet. Viszonyításképpen, egy tipikus bekezdés körülbelül 100-150 tokent tartalmazhat, míg egy teljes könyv több tíz- vagy százezret.

> Léteznek már olyan hatékony módszerek is a Transformer-alapú Nagy Nyelvi Modellek (LLM-ek) skálázására, amelyek végtelenül hosszú bemeneteket tudnak kezelni korlátozott memória- és számításigénnyel.

Miért fontos a kontextusméret?

A nyelvi modell kontextusmérete jelentősen befolyásolja annak képességét a szövegek megértésében és összefüggő, kontextuálisan releváns szövegek generálásában. Íme néhány kulcsfontosságú ok, amiért a kontextusméret számít:

1. **Hosszú szövegek megértése**: A nagyobb kontextusmérettel rendelkező modellek jobban képesek megérteni és elemezni a hosszabb szövegeket, például cikkeket, jelentéseket vagy akár teljes könyveket. Ez kulcsfontosságú olyan feladatoknál, mint a dokumentumok összefoglalása, kérdések megválaszolása és tartalomelemzés.

2. **Koherencia fenntartása**: A nagyobb kontextusablak lehetővé teszi a modell számára, hogy hosszabb szövegrészeken keresztül is fenntartsa a koherenciát és konzisztenciát. Ez fontos olyan feladatoknál, mint a történetgenerálás, párbeszédrendszerek és tartalomkészítés, ahol elengedhetetlen az egységes narratíva vagy téma fenntartása. Ez különösen kritikus, amikor LLM-eket használunk strukturált adatok generálására vagy átalakítására.
3. **Távoli összefüggések felismerése**: Egyes nyelvi feladatok megkövetelik a szövegben egymástól távol eső szavak vagy kifejezések közötti kapcsolatok megértését. A nagyobb kontextusmérettel rendelkező modellek jobban felismerik ezeket a távoli összefüggéseket, ami fontos lehet olyan feladatoknál, mint az érzelemelemzés, fordítás és nyelvi megértés.

4. **Komplex utasítások kezelése**: Olyan alkalmazásoknál, ahol a nyelvi modelleket összetett, többlépéses utasítások követésére használják, a nagyobb kontextusméret lehetővé teszi, hogy a modell a válasz generálásakor a teljes utasításkészletet figyelembe vegye, nem csak a legutóbbi néhány szót.

Példák különböző kontextusméretű nyelvi modellekre

Íme néhány példa különböző kontextusméretű nyelvi modellekre:

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

Mint látható, széles skálán mozog ezeknek a modelleknek a kontextusmérete, az OpenAI GPT-3.5 Turbo modell körülbelül 4 000 tokenjétől az Anthropic Claude v2 modell 200 000 tokenjéig. Néhány modell, mint a Google PaLM 2 és az OpenAI GPT-4, különböző változatokban érhető el nagyobb kontextusméretekkel (pl. "32k" verziók), amelyek még hosszabb bemeneti szekvenciákat tudnak kezelni. És jelenleg (2024 áprilisában) a Google Gemini Pro közel 3 millió tokennel büszkélkedhet!

Érdemes megjegyezni, hogy a kontextusméret változhat egy adott modell konkrét implementációjától és verziójától függően. Például az eredeti OpenAI GPT-4 modell kontextusmérete 8 191 token, míg a későbbi GPT-4 változatok, mint a Turbo és a 4o, jóval nagyobb, 128 000 tokenes kontextusmérettel rendelkeznek.

> Sam Altman a jelenlegi kontextuskorlátokat a személyi számítógépes programozók 80-as évekbeli kilobájtos munkamemória-korlátaihoz hasonlította, és azt mondta, hogy a közeljövőben képesek leszünk “minden személyes adatunkat” beilleszteni egy nagy nyelvi modell kontextusába.

A megfelelő kontextusméret kiválasztása

Amikor egy adott alkalmazáshoz nyelvi modellt választunk, fontos figyelembe venni az adott feladat kontextusméret-követelményeit. Olyan feladatokhoz, amelyek rövid, elszigetelt szövegrészeket tartalmaznak, mint például az érzelemelemzés vagy egyszerű kérdések megválaszolása, egy kisebb kontextusméret is elegendő lehet. Azonban olyan feladatokhoz, amelyek hosszabb, összetettebb szövegek megértését és generálását igénylik, valószínűleg nagyobb kontextusméret szükséges.

Érdemes megjegyezni, hogy a nagyobb kontextusméret gyakran megnövekedett számítási költségekkel és lassabb feldolgozási idővel jár, mivel a modellnek több információt kell figyelembe vennie a válasz generálásakor. Ezért egyensúlyt kell teremteni a kontextusméret és a teljesítmény között az alkalmazásához választott nyelvi modell kiválasztásakor.

> Miért ne válasszuk egyszerűen a legnagyobb kontextusméretű modellt, és töltsük meg annyi információval, amennyivel csak lehet? Nos, a teljesítménytényezőkön kívül a másik fő szempont a költség. 2024 márciusában egyetlen kérés-válasz ciklus a Google Gemini Pro 1.5 teljes kontextussal közel 8 dollárba (USD) kerül. Ha van olyan felhasználási eseted, ami indokolja ezt a költséget, csak rajta! De a legtöbb alkalmazás esetében ez nagyságrendekkel túl drága.

Tűk keresése a szénakazalban

A tű megkeresése a szénakazalban régóta metaforája a nagy adathalmazokban történő visszakeresés kihívásainak. A nagy nyelvi modellek (NNM-ek) területén ezt a hasonlatot kicsit átformáljuk. Képzeljük el, hogy nem csupán egyetlen tényt keresünk egy hatalmas szövegben (mint például Paul Graham esszéinek teljes gyűjteményében), hanem több, szétszórt tényt. Ez a forgatókönyv inkább hasonlít több tű kereséséhez egy hatalmas mezőn, nem csak egyetlen szénakazalban. És itt jön a csavar: nem csak meg kell találnunk ezeket a tűket, hanem össze is kell őket fűznünk egy értelmes szállá.

Amikor az NNM-eknek hosszú szövegkörnyezetbe ágyazott többszörös tények visszakeresésével és értelmezésével kell megbirkózniuk, kettős kihívással szembesülnek. Először is, ott van a visszakeresés pontosságának egyértelmű problémája – ez természetesen csökken, ahogy a tények száma növekszik. Ez várható; elvégre még a legkifinomultabb modelleket is megterheli a szerteágazó szövegben található többszörös részletek nyomon követése.

Másodszor, és talán még kritikusabb a tényekkel való következtetés kihívása. Más dolog tényeket kiválogatni, és megint más őket koherens narratívává vagy válasszá szintetizálni. Itt jön a valódi próbatétel. Az NNM-ek teljesítménye a következtetési feladatokban általában még jobban romlik, mint az egyszerű visszakeresési feladatokban. Ez a romlás nem csak a mennyiségről szól; ez a kontextus, relevancia és következtetés bonyolult táncáról szól.

Miért történik ez? Nos, vegyük figyelembe az emberi megismerés memória- és figyelemdinamikáját, amely bizonyos mértékig tükröződik az NNM-ekben. Nagy mennyiségű információ feldolgozásakor az NNM-ek, akárcsak az emberek, elveszíthetik a korábbi részletek fonalát, miközben újakat fogadnak be. Ez különösen igaz azokra a modellekre, amelyeket nem kifejezetten úgy terveztek, hogy automatikusan priorizálják vagy újra megvizsgálják a szöveg korábbi részeit.

Ráadásul egy NNM azon képessége, hogy ezeket a visszakeresett tényeket koherens válasszá szője, hasonlít a narratívaépítéshez. Ez nem csupán az információ

visszakeresését igényli, hanem mély megértést és kontextuális elhelyezést is, ami továbbra is komoly kihívást jelent a jelenlegi MI számára.

Tehát mit jelent ez számunkra, e technológiák fejlesztői és integrálói számára? Tisztában kell lennünk ezekkel a korlátokkal, amikor olyan rendszereket tervezünk, amelyek NNM-ekre támaszkodnak összetett, hosszú szöveges feladatok kezelésében. Annak megértése, hogy a teljesítmény bizonyos körülmények között romolhat, segít reális elvárások kialakításában és jobb tartalékmechanizmusok vagy kiegészítő stratégiák kidolgozásában.

Modalitások: A szövegen túl

Bár a mai nyelvi modellek többsége a szöveg feldolgozására és generálására összpontosít, egyre erősödik a többmodalitású modellek felé mutató trend, amelyek természetes módon képesek többféle adattípust, például képeket, hangot és videót feldolgozni és létrehozni. Ezek a többmodalitású modellek új lehetőségeket nyitnak meg olyan MI-alapú alkalmazások számára, amelyek különböző modalitásokban képesek megérteni és generálni tartalmakat.

Mik azok a modalitások?

A nyelvi modellek kontextusában a modalitások azokra a különböző adattípusokra utalnak, amelyeket egy modell feldolgozni és generálni képes. A leggyakoribb modalitás a szöveg, amely magában foglalja az írott nyelv különböző formáit, mint például könyveket, cikkeket, weboldalakat és közösségi média bejegyzéseket. Azonban több más modalitást is egyre inkább beépítenek a nyelvi modellekbe:

- **Képek**: Vizuális adatok, mint például fényképek, illusztrációk és diagramok.
- **Hang**: Hangadatok, mint például beszéd, zene és környezeti hangok.
- **Videó**: Mozgó vizuális adatok, gyakran hanggal kísérve, mint például videoklipek és filmek.

Minden modalitás egyedi kihívásokat és lehetőségeket jelent a nyelvi modellek számára. Például a képek esetében a modellnek meg kell értenie a vizuális koncepciókat és kapcsolatokat, míg a hang esetében a beszédet és egyéb hangokat kell feldolgoznia és generálnia.

Többmodalitású nyelvi modellek

A többmodalitású nyelvi modellek úgy vannak kialakítva, hogy egyetlen modellen belül több modalitást is kezeljenek. Ezek a modellek általában specializált komponensekkel vagy rétegekkel rendelkeznek, amelyek képesek mind megérteni a bemeneti adatokat, mind kimeneti adatokat generálni különböző modalitásokban. Néhány figyelemre méltó példa a többmodalitású nyelvi modellekre:

- **OpenAI GPT-4o**: A GPT-4o egy olyan nagy nyelvi modell, amely a szöveg mellett természetes módon érti és feldolgozza a beszédhangot is. Ez a képesség lehetővé teszi olyan feladatok elvégzését, mint a beszélt nyelv átírása, szöveg generálása hangbemenetből és beszédalapú kérdésekre való válaszadás.
- **OpenAI GPT-4 vizuális bemenettel**: A GPT-4 egy olyan nagy nyelvi modell, amely képes mind szöveget, mind képeket feldolgozni. Amikor képet kap bemenetként, a GPT-4 elemezni tudja a kép tartalmát, és olyan szöveget tud generálni, amely leírja vagy reagál a vizuális információra.
- **Google Gemini**: A Gemini egy többmodalitású modell, amely képes kezelni szöveget, képeket és videót. Egységes architektúrát használ, amely lehetővé teszi a modalitások közötti megértést és generálást, olyan feladatok elvégzését téve lehetővé, mint a képfeliratozás, videóösszefoglalás és vizuális kérdésmegválaszolás.

- **DALL-E és Stable Diffusion**: Bár nem hagyományos értelemben vett nyelvi modellek, ezek a modellek a többmodális MI erejét demonstrálják azáltal, hogy szöveges leírásokból képeket generálnak. Szemléltetik azoknak a modelleknek a potenciálját, amelyek különböző modalitások között képesek fordítani.

A többmodális modellek előnyei és alkalmazásai

A többmodális nyelvi modellek számos előnyt kínálnak és sokféle alkalmazást tesznek lehetővé, többek között:

- **Fejlett megértés**: Több modalitásból származó információ feldolgozásával ezek a modellek átfogóbb világértelmezésre képesek, hasonlóan ahhoz, ahogyan az emberek különböző érzékszervi bemenetekből tanulnak.
- **Keresztmodális generálás**: A többmodális modellek képesek egy modalitásban tartalmat generálni egy másik modalitásból származó bemenet alapján, például szöveges leírásból képet alkotni vagy írásos cikkből videós összefoglalót készíteni.
- **Hozzáférhetőség**: A többmodális modellek az információt hozzáférhetőbbé tehetik a modalitások közötti fordítással, például képekről szöveges leírást generálva látássérült felhasználók számára, vagy írásos tartalmakból hangváltozatot készítve.
- **Kreatív alkalmazások**: A többmodális modellek használhatók kreatív feladatokra, például művészet, zene vagy videók létrehozására szöveges utasítások alapján, új lehetőségeket nyitva művészek és tartalomalkotók számára.

Ahogy a többmodális nyelvi modellek továbbfejlődnek, valószínűleg egyre fontosabb szerepet játszanak majd az olyan MI-alapú alkalmazások fejlesztésében, amelyek képesek több modalitásban megérteni és generálni tartalmakat. Ez lehetővé teszi a természetesebb és intuitívabb interakciókat az emberek és az MI-rendszerek között, valamint új lehetőségeket nyit meg a kreatív kifejezés és a tudás terjesztése terén.

Szolgáltatói ökoszisztémák

Amikor nagy nyelvi modellek (NNM-ek) alkalmazásokba való beépítéséről van szó, egyre több lehetőség közül választhatunk. Minden jelentős NNM-szolgáltató, mint például az OpenAI, az Anthropic, a Google és a Cohere, saját ökoszisztémát kínál

modellekből, API-kból és eszközökből. A megfelelő szolgáltató kiválasztása során különböző tényezőket kell figyelembe venni, beleértve az árazást, teljesítményt, tartalomszűrést, adatvédelmet és testreszabási lehetőségeket.

OpenAI

Az OpenAI az NNM-ek egyik legismertebb szolgáltatója, GPT sorozatával (GPT-3, GPT-4), amelyet széles körben használnak különböző alkalmazásokban. Az OpenAI felhasználóbarát API-t kínál, amely lehetővé teszi modelleik egyszerű integrálását alkalmazásokba. Különböző képességű és árpontú modellek választékát kínálják, a kezdő szintű Ada modelltől a nagy teljesítményű Davinci modellig.

Az OpenAI ökoszisztémája olyan eszközöket is tartalmaz, mint az OpenAI Playground, amely lehetővé teszi a promptokkal való kísérletezést és a modellek finomhangolását specifikus használati esetekhez. Tartalomszűrési opciókat is kínálnak a nem megfelelő vagy káros tartalom generálásának megakadályozására.

Az OpenAI modellek közvetlen használatakor Alex Rudall ruby-openai könyvtárára támaszkodom.

Anthropic

Az Anthropic egy másik jelentős szereplő az NNM térben, Claude modelljeik egyre népszerűbbek erős teljesítményük és etikai megfontolásaik miatt. Az Anthropic a biztonságos és felelősségteljes MI-rendszerek fejlesztésére összpontosít, erős hangsúlyt fektetve a tartalomszűrésre és a káros kimenetek elkerülésére.

Az Anthropic ökoszisztémája magában foglalja a Claude API-t, amely lehetővé teszi a modell integrálását alkalmazásaikba, valamint eszközöket kínál promptmérnökséghez és finomhangoláshoz. Kínálják a Claude Instant modellt is, amely webes keresési képességekkel rendelkezik a naprakészebb és tényszerűbb válaszok érdekében.

Az Anthropic modellek közvetlen használatakor Alex Rudall anthrophic könyvtárára támaszkodom.

Google

A Google több nagy teljesítményű NNM-et fejlesztett ki, köztük a Geminit, BERT-et, T5-öt és PaLM-ot. Ezek a modellek széles körű természetes nyelvfeldolgozási feladatokban nyújtott erős teljesítményükről ismertek. A Google ökoszisztémája magában foglalja a TensorFlow és Keras könyvtárakat, amelyek eszközöket és keretrendszereket biztosítanak gépi tanulási modellek építéséhez és tanításához.

A Google Cloud AI Platform szolgáltatást is kínál, amely lehetővé teszi modelleik egyszerű telepítését és skálázását a felhőben. Előre betanított modellek és API-k széles választékát kínálják olyan feladatokhoz, mint az érzelemelemzés, entitásfelismerés és fordítás.

Meta

A Meta, korábban Facebook, mélyen elkötelezett a nagy nyelvi modellek fejlesztése mellett, amit olyan modellek kiadása is jelez, mint a LLaMA és az OPT. Ezek a modellek kiemelkednek a különböző nyelvi feladatokban nyújtott erős teljesítményükkel, és nagyrészt nyílt forráskódú csatornákon keresztül érhetők el, támogatva a Meta elkötelezettségét a kutatás és közösségi együttműködés mellett.

A Meta ökoszisztémája elsősorban a PyTorch köré épül, amely egy nyílt forráskódú gépi tanulási könyvtár, amelyet dinamikus számítási képességei és rugalmassága miatt kedvelnek, elősegítve az innovatív MI-kutatást és -fejlesztést.

A technikai megoldásaik mellett a Meta nagy hangsúlyt fektet az etikus MI-fejlesztésre. Robusztus tartalomszűrést alkalmaznak és az elfogultságok csökkentésére összpontosítanak, összhangban az MI-alkalmazások biztonságára és felelősségteljes használatára vonatkozó átfogó céljaikkal.

Cohere

A Cohere egy újabb szereplő az LLM területén, amely arra összpontosít, hogy a nagyméretű nyelvi modelleket könnyebben hozzáférhetővé és használhatóbbá tegye,

mint versenytársai. Ökoszisztémájuk magában foglalja a Cohere API-t, amely különféle előre betanított modellekhez biztosít hozzáférést olyan feladatokhoz, mint a szöveggenerálás, osztályozás és összegzés.

A Cohere eszközöket kínál prompttervezéshez, finomhangoláshoz és tartalomszűréshez is. Nagy hangsúlyt fektetnek az adatvédelemre és biztonságra, olyan funkciókkal, mint a titkosított adattárolás és hozzáférés-vezérlés.

Ollama

Az Ollama egy saját üzemeltetésű platform, amely lehetővé teszi a felhasználók számára különböző nagyméretű nyelvi modellek (LLM-ek) helyi kezelését és telepítését saját gépeiken, teljes ellenőrzést biztosítva az MI-modelljeik felett, külső felhőszolgáltatások használata nélkül. Ez a konfiguráció ideális azok számára, akik előnyben részesítik az adatvédelmet, és házon belül szeretnék kezelni MI-műveleteiket.

A platform számos modellt támogat, beleértve a Llama, Phi, Gemma és Mistral különböző verzióit, amelyek méretben és számítási követelményekben eltérnek. Az Ollama megkönnyíti ezeknek a modelleknek a letöltését és futtatását közvetlenül a parancssorból olyan egyszerű parancsokkal, mint az `ollama run <model_name>`, és úgy tervezték, hogy különböző operációs rendszereken működjön, beleértve a macOS-t, Linuxot és Windowst.

Azon fejlesztők számára, akik nyílt forráskódú modelleket szeretnének integrálni alkalmazásaikba távoli API használata nélkül, az Ollama CLI-t kínál a modellek életciklusának kezeléséhez, hasonlóan a konténerkezelő eszközökhöz. Támogatja az egyéni konfigurációkat és promptokat is, lehetővé téve a nagyfokú testreszabást a modellek specifikus igényekhez vagy felhasználási esetekhez való igazításához.

Az Ollama különösen alkalmas a technológiában jártas felhasználók és fejlesztők számára a parancssori felülete és a rugalmassága miatt, amit az MI-modellek kezelésében és telepítésében nyújt. Ez hatékony eszközzé teszi olyan vállalkozások

és magánszemélyek számára, akiknek robusztus MI-képességekre van szükségük a biztonság és az irányítás feladása nélkül.

Többmodelles Platformok

Emellett vannak olyan szolgáltatók, amelyek számos nyílt forráskódú modellt üzemeltetnek, mint például a Together.ai és a Groq. Ezek a platformok rugalmasságot és testreszabhatóságot kínálnak, lehetővé téve a nyílt forráskódú modellek futtatását, és bizonyos esetekben még a finomhangolását is az egyedi igényeknek megfelelően. Például a Together.ai hozzáférést biztosít számos nyílt forráskódú LLM-hez, lehetővé téve a felhasználók számára különböző modellek és konfigurációk kipróbálását. A Groq olyan ultra nagy teljesítményű szövegkiegészítésre összpontosít, amely e könyv írásakor szinte varázslatosnak tűnik

LLM-szolgáltató választása

LLM-szolgáltató választásakor a következő tényezőket érdemes figyelembe venni:

- **Árazás**: A különböző szolgáltatók eltérő árazási modelleket kínálnak, a használat alapú fizetéstől kezdve az előfizetéses csomagokig. Fontos figyelembe venni a várható használatot és a költségvetést a szolgáltató kiválasztásakor.
- **Teljesítmény**: Az LLM-ek teljesítménye jelentősen változhat a szolgáltatók között, ezért fontos a modellek tesztelése és teljesítménymérése konkrét használati esetekben a döntés előtt.
- **Tartalomszűrés**: Az alkalmazástól függően a tartalomszűrés kritikus szempont lehet. Egyes szolgáltatók robusztusabb tartalomszűrési lehetőségeket kínálnak másoknál.
- **Adatvédelem**: Ha az alkalmazás érzékeny felhasználói adatokat kezel, fontos olyan szolgáltatót választani, amely erős adatvédelmi és biztonsági gyakorlatokkal rendelkezik.

- **Testreszabás**: Egyes szolgáltatók nagyobb rugalmasságot kínálnak a modellek finomhangolásában és testreszabásában specifikus használati esetekhez.

Végső soron az LLM-szolgáltató kiválasztása az alkalmazás specifikus követelményeitől és korlátaitól függ. Az opciók gondos értékelésével és olyan tényezők figyelembevételével, mint az árazás, teljesítmény és adatvédelem, kiválasztható az igényeknek legjobban megfelelő szolgáltató.

Érdemes megjegyezni, hogy az LLM-ek területe folyamatosan fejlődik, rendszeresen jelennek meg új szolgáltatók és modellek. Érdemes naprakésznek maradni a legújabb fejleményekkel kapcsolatban és nyitottnak lenni az új lehetőségek felfedezésére, ahogy azok elérhetővé válnak.

OpenRouter

Ebben a könyvben kizárólag az OpenRouter szolgáltatót fogom használni API-szolgáltatóként. Az ok egyszerű: ez egy egyablakos megoldás az összes legnépszerűbb kereskedelmi és nyílt forráskódú modellhez. Ha alig várja, hogy belekezdhessen az MI-programozásba, az egyik legjobb kiindulópont a saját OpenRouter Ruby könyvtáram.

A teljesítmény átgondolása

A nyelvi modellek alkalmazásokba történő beépítésekor a teljesítmény kulcsfontosságú szempont. Egy nyelvi modell teljesítménye mérhető a *késleltetés* (a válasz generálásához szükséges idő) és az *áteresztőképesség* (az időegység alatt kezelhető kérések száma) szempontjából.

Az *első token megjelenéséig eltelt idő* (TTFT) egy másik alapvető teljesítménymutató, amely különösen fontos a chatbotok és az interaktív, valós idejű válaszokat igénylő alkalmazások esetében. A TTFT azt az időt méri, ami a felhasználói kérés beérkezésétől

a válasz első szavának (vagy tokenjének) generálásáig telik el. Ez a mutató kulcsfontosságú a zökkenőmentes és lebilincselő felhasználói élmény fenntartásához, mivel a késleltetett válaszok a felhasználók frusztrációjához és elfordulásához vezethetnek.

Ezek a teljesítménymutatók jelentős hatással lehetnek a felhasználói élményre és az alkalmazás skálázhatóságára.

Számos tényező befolyásolhatja egy nyelvi modell teljesítményét, többek között:

Paraméterszám: A több paraméterrel rendelkező nagyobb modellek általában több számítási erőforrást igényelnek, és magasabb késleltetéssel, valamint alacsonyabb áteresztőképességgel rendelkezhetnek a kisebb modellekhez képest.

Hardver: Egy nyelvi modell teljesítménye jelentősen változhat attól függően, hogy milyen hardveren fut. A felhőszolgáltatók gépi tanulási munkaterhelésre optimalizált GPU és TPU példányokat kínálnak, amelyek nagymértékben felgyorsíthatják a modell következtetési folyamatát.

Az OpenRouter egyik előnye, hogy számos általa kínált modell esetében választhatunk a felhőszolgáltatók közül, amelyek különböző teljesítményprofilokat és költségeket kínálnak.

Kvantálás: A kvantálási technikák használhatók a modell memóriaigényének és számítási követelményeinek csökkentésére azáltal, hogy a súlyokat és aktiválásokat alacsonyabb pontosságú adattípusokkal reprezentálják. Ez javíthatja a teljesítményt anélkül, hogy jelentősen rontaná a minőséget. Alkalmazásfejlesztőként valószínűleg nem fog saját modelleket különböző kvantálási szinteken betanítani, de jó, ha legalább ismeri a terminológiát.

Kötegelt feldolgozás: Több kérés egyidejű feldolgozása kötegekben javíthatja az áteresztőképességet a modell betöltésének és az adatátvitel többletköltségének amortizálásával.

Gyorsítótárazás: A gyakran használt promptok vagy bemeneti szekvenciák eredményeinek gyorsítótárazása csökkentheti a következtetési kérések számát és javíthatja az általános teljesítményt.

Amikor nyelvi modellt választunk egy éles alkalmazáshoz, fontos a teljesítmény tesztelése reprezentatív munkaterhelések és hardverkonfigurációk mellett. Ez segíthet azonosítani a potenciális szűk keresztmetszeteket és biztosítani, hogy a modell megfeleljen az elvárt teljesítménycéloknak.

Érdemes megfontolni a modell teljesítménye és egyéb tényezők, mint például a költség, rugalmasság és integrációs könnyűség közötti kompromisszumokat. Például egy kisebb, olcsóbb modell használata alacsonyabb késleltetéssel előnyösebb lehet olyan alkalmazásoknál, amelyek valós idejű válaszokat igényelnek, míg egy nagyobb, erőteljesebb modell jobban megfelelhet kötegelt feldolgozáshoz vagy összetett következtetési feladatokhoz.

Kísérletezés különböző LLM modellekkel

Egy LLM kiválasztása ritkán végleges döntés. Mivel rendszeresen jelennek meg új és fejlettebb modellek, érdemes az alkalmazásokat moduláris módon felépíteni, ami lehetővé teszi különböző nyelvi modellek cseréjét az idő múlásával. A promptok és adatkészletek gyakran minimális változtatásokkal újrafelhasználhatók különböző modellek között. Ez lehetővé teszi, hogy kihasználjuk a nyelvi modellezés legújabb fejlesztéseit anélkül, hogy teljesen át kellene terveznünk az alkalmazásainkat.

A széles modellválaszték közötti egyszerű váltás lehetősége egy újabb ok, amiért szeretem az OpenRoutert.

Amikor új nyelvi modellre váltunk, fontos alaposan tesztelni és validálni annak teljesítményét és kimeneti minőségét, hogy megbizonyosodjunk arról, megfelel-e az alkalmazás követelményeinek. Ez magában foglalhatja a modell újratanítását vagy

finomhangolását domain-specifikus adatokon, valamint a modell kimeneteitől függő downstream komponensek frissítését.

Ha az alkalmazásokat a teljesítményt és modularitást szem előtt tartva tervezzük, skálázható, hatékony és jövőbiztos rendszereket hozhatunk létre, amelyek képesek alkalmazkodni a nyelvi modellezési technológia gyorsan fejlődő világához.

Összetett MI-rendszerek

Mielőtt lezárnánk a bevezetőnket, érdemes megemlíteni, hogy 2023 előtt és a ChatGPT által kiváltott generatív MI iránti érdeklődés robbanása előtt a hagyományos MI-megközelítések általában egyetlen, zárt modell integrációjára támaszkodtak. Ezzel szemben az *Összetett MI-rendszerek* összekapcsolt komponensek komplex folyamatait használják az intelligens viselkedés elérése érdekében.

Az összetett MI-rendszerek lényegében több modulból állnak, amelyek mindegyike specifikus feladatok vagy funkciók végrehajtására készült. Ezek a modulok tartalmazhatnak generátorokat, lekérdezőket, rangsorolókat, osztályozókat és különböző egyéb specializált komponenseket. A teljes rendszer kisebb, fókuszált egységekre bontásával a fejlesztők rugalmasabb, skálázhatóbb és karbantarthatóbb MI-architektúrákat hozhatnak létre.

Az összetett MI-rendszerek egyik kulcsfontosságú előnye, hogy képesek ötvözni a különböző MI-technikák és modellek erősségeit. Például egy rendszer használhat nagy nyelvi modellt (LLM) a természetes nyelvértéshez és -generáláshoz, miközben egy külön modellt alkalmaz információ-visszakeresésre vagy szabályalapú döntéshozatalra. Ez a moduláris megközelítés lehetővé teszi, hogy minden konkrét feladathoz a legjobb eszközöket és technikákat válasszuk ki, ahelyett, hogy egyetlen univerzális megoldásra hagyatkoznánk.

Azonban az összetett MI-rendszerek építése egyedi kihívásokat is jelent. Különösen a rendszer viselkedésének általános koherenciája és konzisztenciája igényel robusztus

tesztelési, monitorozási és irányítási mechanizmusokat.

A GPT-4 és hasonló nagy teljesítményű LLM-ek megjelenése minden eddiginél könnyebbé teszi az összetett MI-rendszerekkel való kísérletezést, mivel ezek a fejlett modellek képesek többféle szerepet is betölteni egy összetett rendszeren belül, mint például osztályozás, rangsorolás és generálás, a természetes nyelvértési képességeik mellett. Ez a sokoldalúság lehetővé teszi a fejlesztők számára, hogy gyorsan prototípusokat készítsenek és iteráljanak az összetett MI-architektúrákon, új lehetőségeket nyitva az intelligens alkalmazásfejlesztés terén.

Összetett MI-rendszerek telepítési mintái

Az összetett MI-rendszerek különböző minták szerint telepíthetők, amelyek mindegyikét specifikus követelmények és használati esetek kiszolgálására tervezték. Vizsgáljunk meg négy gyakori telepítési mintát: Kérdés és válasz, Többügynökös/Ágens alapú problémamegoldók, Beszélgető MI és CoPilotok.

Kérdés és válasz

A Kérdés és válasz (Q&A) rendszerek olyan információ-visszakeresésre összpontosítanak, amelyet MI-modellek értelmezési képességeivel bővítettek, hogy többet nyújtsanak egy egyszerű keresőmotornál. A nagy nyelvi modellek és külső tudásforrások kombinálásával, Visszakereséssel bővített generálás (RAG) segítségével, a Kérdés és válasz rendszerek elkerülik a hallucinációkat és pontos, kontextuálisan releváns válaszokat adnak a felhasználói kérdésekre.

Egy LLM-alapú Q&A rendszer kulcsfontosságú komponensei:

- **Lekérdezés-értelmezés és -átfogalmazás**: A felhasználói kérdések elemzése és átfogalmazása, hogy jobban illeszkedjenek az alapul szolgáló tudásforrásokhoz.

- **Tudás-visszakeresés**: Releváns információk visszakeresése strukturált vagy strukturálatlan adatforrásokból az átfogalmazott kérdés alapján.
- **Válaszgenerálás**: Koherens és informatív válaszok generálása a visszakeresett tudás és a nyelvi modell generatív képességeinek integrálásával.

A RAG alrendszerek különösen fontosak olyan Q&A területeken, ahol kulcsfontosságú a pontos és naprakész információk szolgáltatása, mint például az ügyfélszolgálat, tudásmenedzsment vagy oktatási alkalmazások.

Többügynökös/Ágens alapú problémamegoldók

A többügynökös, más néven *ágens alapú* rendszerek több autonóm ügynökből állnak, amelyek együttműködnek komplex problémák megoldásában. Minden ügynöknek megvan a saját szerepe, készségkészlete és hozzáférése a releváns eszközökhöz vagy információforrásokhoz. Az együttműködés és információcsere révén ezek az ügynökök olyan feladatokat is meg tudnak oldani, amelyekkel egyetlen ügynök egyedül nem boldogulna.

A többügynökös problémamegoldók kulcsfontosságú elvei:

- **Specializáció**: Minden ügynök a probléma egy specifikus aspektusára összpontosít, kihasználva egyedi képességeit és tudását.
- **Együttműködés**: Az ügynökök kommunikálnak és koordinálják tevékenységeiket egy közös cél elérése érdekében, gyakran üzenetküldésen vagy megosztott memórián keresztül.
- **Alkalmazkodóképesség**: A rendszer képes alkalmazkodni a változó feltételekhez vagy követelményekhez az egyes ügynökök szerepeinek és viselkedésének módosításával.

A többügynökös rendszerek jól alkalmazhatók olyan területeken, amelyek elosztott problémamegoldást igényelnek, mint például az ellátási lánc optimalizálása, forgalomirányítás vagy vészhelyzeti reagálás tervezése.

Beszélgető MI

A beszélgető MI-rendszerek természetes nyelvi interakciókat tesznek lehetővé a felhasználók és az intelligens ügynökök között. Ezek a rendszerek ötvözik a természetes nyelvértést, párbeszédkezelést és nyelvgenerálást, hogy lebilincselő és személyre szabott beszélgetési élményt nyújtsanak.

Egy beszélgető MI-rendszer fő komponensei:

- **Szándékfelismerés**: A felhasználó szándékának azonosítása a bemenet alapján, például kérdésfeltevés, kérés vagy vélemény kifejezése.
- **Entitáskinyerés**: Releváns entitások vagy paraméterek kinyerése a felhasználói bemenetből, például dátumok, helyek vagy terméknevek.
- **Párbeszédkezelés**: A beszélgetés állapotának fenntartása, a megfelelő válasz meghatározása a felhasználó szándéka és a kontextus alapján, valamint a többfordulós interakciók kezelése.
- **Válaszgenerálás**: Emberszerű válaszok generálása nyelvi modellek, sablonok vagy visszakeresésen alapuló módszerek segítségével.

A beszélgető MI-rendszereket gyakran alkalmazzák ügyfélszolgálati chatbotokban, virtuális asszisztensekben és hangvezérelt felületekben. Ahogy korábban említettük, a könyvben szereplő megközelítések, minták és kódpéldák többsége közvetlenül az Olympia nevű nagy beszélgető MI-rendszeren végzett munkából származik.

CoPilotok

A CoPilotok olyan MI-alapú asszisztensek, amelyek az emberi felhasználókkal együttműködve növelik azok produktivitását és döntéshozatali képességeit. Ezek a rendszerek a természetes nyelvfeldolgozás, a gépi tanulás és a szakterület-specifikus tudás kombinációját használják fel az intelligens ajánlások készítéséhez, feladatok automatizálásához és kontextuális támogatás nyújtásához.

A CoPilotok főbb jellemzői:

- **Személyre szabás**: Alkalmazkodás az egyéni felhasználói preferenciákhoz, munkafolyamatokhoz és kommunikációs stílusokhoz.
- **Proaktív segítségnyújtás**: A felhasználói igények előrejelzése és releváns javaslatok vagy műveletek felajánlása kifejezett kérés nélkül.
- **Folyamatos tanulás**: A teljesítmény idővel történő javítása a felhasználói visszajelzések, interakciók és adatok alapján.

A CoPilotokat egyre több területen alkalmazzák, például szoftverfejlesztésben (pl. kódkiegészítés és hibakeresés), kreatív írásban (pl. tartalmi javaslatok és szerkesztés), és adatelemzésben (pl. betekintések és vizualizációs javaslatok)

Ezek az alkalmazási minták jól szemléltetik az összetett MI-rendszerek sokoldalúságát és potenciálját. Az egyes minták jellemzőinek és felhasználási eseteinek megértésével megalapozott döntéseket hozhat az intelligens alkalmazások tervezése és megvalósítása során. Bár ez a könyv nem kifejezetten az összetett MI-rendszerek megvalósításáról szól, ugyanezek a megközelítések és minták sok, ha nem minden esetben alkalmazhatók különálló MI-komponensek integrálására a hagyományos alkalmazásfejlesztésen belül.

Szerepkörök az összetett MI-rendszerekben

Az összetett MI-rendszerek összekapcsolt modulok alapjára épülnek, amelyek mindegyike egy adott szerep betöltésére lett tervezve. Ezek a modulok együttműködnek az intelligens viselkedés létrehozása és a komplex problémák megoldása érdekében. Hasznos ismerni ezeket a szerepköröket, amikor arról gondolkodunk, hogy hol tudnánk alkalmazásunk részeit MI-komponensekkel megvalósítani vagy helyettesíteni.

Generátor

A generátorok feladata új adatok vagy tartalom előállítása tanult minták vagy bemeneti promptok alapján. Az MI világában sokféle generátor létezik, de a könyvben bemutatott nyelvi modellek kontextusában a generátorok képesek emberszerű szöveget létrehozni,

befejezetlen mondatokat kiegészíteni vagy felhasználói kérdésekre válaszokat generálni. Kulcsszerepet játszanak olyan feladatokban, mint a tartalomgenerálás, párbeszédgenerálás és adatbővítés.

Lekérdező

A lekérdezőket nagy adathalmazokból vagy tudásbázisokból való információkeresésre és -kinyerésre használják. Olyan technikákat alkalmaznak, mint a szemantikus keresés, kulcsszóillesztés vagy vektoros hasonlóság, hogy megtalálják a legmegfelelőbb adatpontokat egy adott lekérdezés vagy kontextus alapján. A lekérdezők elengedhetetlenek olyan feladatokhoz, amelyek gyors hozzáférést igényelnek specifikus információkhoz, mint például kérdésmegválaszolás, tényellenőrzés vagy tartalomajánlás.

Rangsoroló

A rangsorolók feladata elemek rendezése vagy prioritizálása meghatározott kritériumok vagy relevanciaszámok alapján. Súlyokat vagy pontszámokat rendelnek minden elemhez, majd ezek alapján sorba rendezik őket. A rangsorolókat gyakran használják keresőmotorokban, ajánlórendszerekben vagy bármely olyan alkalmazásban, ahol kritikus fontosságú a legrelevánsabb eredmények megjelenítése a felhasználók számára.

Osztályozó

Az osztályozókat adatpontok előre meghatározott osztályokba vagy kategóriákba sorolására használják. Címkézett tanítóadatokból tanulnak, majd új, korábban nem látott példányok osztályát jósolják meg. Az osztályozók alapvető fontosságúak olyan feladatokban, mint a hangulatelem

zés, spam-felismerés vagy képfelismerés, ahol a cél minden bemenethez egy specifikus kategória hozzárendelése.

Eszközök és Ágensek

E központi szerepkörök mellett az összetett MI-rendszerek gyakran tartalmaznak eszközöket és ágenseket funkcionalitásuk és alkalmazkodóképességük növelése érdekében:

- **Eszközök**: Az eszközök olyan különálló szoftverkomponensek vagy API-k, amelyek specifikus műveleteket vagy számításokat végeznek. Ezeket más modulok, például generátorok vagy lekérdezők hívhatják meg részfeladatok végrehajtásához vagy további információk gyűjtéséhez. Az eszközök példái közé tartoznak a webes keresőmotorok, számológépek vagy adatvizualizációs könyvtárak.
- **Ágensek**: Az ágensek olyan autonóm entitások, amelyek képesek érzékelni környezetüket, döntéseket hozni és cselekedni specifikus célok elérése érdekében. Gyakran különböző MI-technikák kombinációjára támaszkodnak, mint például tervezés, következtetés és tanulás, hogy hatékonyan működjenek dinamikus vagy bizonytalan körülmények között. Az ágensek használhatók komplex viselkedések modellezésére vagy több modul működésének koordinálására egy összetett MI-rendszeren belül.

Egy tisztán összetett MI-rendszerben e komponensek közötti interakciót jól definiált interfészek és kommunikációs protokollok irányítják. Az adatok a modulok között áramlanak, ahol az egyik komponens kimenete egy másik bemenetéül szolgál. Ez a moduláris architektúra rugalmasságot, skálázhatóságot és karbantarthatóságot biztosít, mivel az egyes komponensek frissíthetők, cserélhetők vagy bővíthetők anélkül, hogy az egész rendszert befolyásolnák.

Ezen komponensek és interakcióik erejét kihasználva az összetett MI-rendszerek képesek olyan komplex, valós problémák kezelésére, amelyek különböző MI-képességek kombinációját igénylik. Ahogy az MI alkalmazásfejlesztésbe való integrálásának

megközelítéseit és mintáit feltárjuk, tartsuk szem előtt, hogy az összetett MI-rendszerekben használt ugyanazon elvek és technikák alkalmazhatók intelligens, adaptív és felhasználóközpontú alkalmazások létrehozására.

Az 1. rész következő fejezeteiben mélyebben megvizsgáljuk az MI-komponensek alkalmazásfejlesztési folyamatba való integrálásának alapvető megközelítéseit és technikáit. A prompt-tervezéstől és a lekérdezéssel bővített generáláson át az öngyógyító adatokig és az intelligens munkafolyamat-vezérlésig számos mintát és bevált gyakorlatot tárgyalunk, amelyek segítenek élvonalbeli MI-alapú alkalmazások építésében.

1. rész: Alapvető Megközelítések és Technikák

A könyv ezen része különböző módszereket mutat be az MI alkalmazásokba történő integrálására. A fejezetek egymással összefüggő megközelítések és technikák sorát tárgyalják, a magasabb szintű koncepcióktól, mint Az Út Szűkítése és a Visszakereséssel Bővített Generálás, egészen az LLM csevegés-kiegészítő API-k fölé saját absztrakciós réteg programozásának ötletéig.

A könyv ezen részének célja, hogy segítsen megérteni azokat a viselkedésformákat, amelyeket MI-vel implementálhatsz, mielőtt túlságosan belemélyednénk a 2. részben tárgyalt specifikus implementációs mintákba.

Az 1. részben szereplő megközelítések olyan ötleteken alapulnak, amelyeket a saját kódomban használtam, klasszikus vállalati alkalmazásarchitektúra és integrációs mintákon, valamint olyan metaforákon, amelyeket az MI képességeinek magyarázására használtam másoknak, beleértve a nem technikai területen dolgozó üzleti érintetteket is.

Az út szűkítése

“Az út szűkítése” arra utal, hogy az MI-t az adott feladatra összpontosítjuk. Ezt használom mantraként, amikor kezdek frusztrált lenni amiatt, hogy az MI “bután” vagy váratlan módon viselkedik. A mantra emlékeztet arra, hogy a hiba valószínűleg az én hibám, és hogy valószínűleg még jobban kellene szűkítenem az utat.

Az út szűkítésének szükségessége a nagy nyelvi modellekben rejlő hatalmas tudásmennyiségből ered, különösen az olyan világszínvonalú modellek esetében, mint az OpenAI és az Anthropic, amelyek szó szerint több billió paraméterrel

rendelkeznek.

Az ilyen széles körű tudáshoz való hozzáférés kétségtelenül hatékony, és olyan emergens viselkedéseket eredményez, mint a tudatelmélet és az emberszerű gondolkodás képessége. Azonban ez a földrengésszerű információmennyiség kihívásokat is jelent, amikor pontos és precíz válaszokat kell generálni specifikus promptokra, különösen akkor, ha ezeknek a promptoknak determinisztikus viselkedést kell mutatniuk, amely integrálható a "normál" szoftverfejlesztéssel és algoritmusokkal.

Számos tényező vezet ezekhez a kihívásokhoz.

Információs túlterhelés: A nagy nyelvi modellek hatalmas mennyiségű adaton tanulnak, amelyek különböző területeket, forrásokat és időszakokat ölelnek fel. Ez a kiterjedt tudás lehetővé teszi számukra, hogy különböző témákban vegyenek részt, és a világ széles körű megértésén alapuló válaszokat generáljanak. Azonban amikor egy konkrét prompttal szembesülnek, a modell küzdhet azzal, hogy kiszűrje az irreleváns, ellentmondásos vagy elavult információkat, ami a fókusz vagy pontosság hiányához vezethet a válaszokban. Attól függően, hogy mit próbálsz elérni, az *ellentmondásos* információk puszta mennyisége könnyen túlterhelheti a modell képességét arra, hogy megadja a kívánt választ vagy viselkedést.

Kontextuális kétértelműség: Tekintettel a tudás hatalmas *látens terére*, a nagy nyelvi modellek kétértelműséggel találkozhatnak, amikor megpróbálják megérteni a promptod *kontextusát*. Megfelelő szűkítés vagy iránymutatás nélkül a modell olyan válaszokat generálhat, amelyek csak érintőlegesen kapcsolódnak, de nem közvetlenül relevánsak a szándékaidhoz. Az ilyen típusú hiba olyan válaszokhoz vezet, amelyek nem a témába vágóak, következetlenek, vagy nem felelnek meg a meghatározott igényeidnek. Ebben az esetben az út szűkítése a kontextus *egyértelműsítésére* utal, biztosítva, hogy az általad megadott kontextus miatt a modell csak az alaptudásában lévő legrelevánsabb információkra összpontosítson.

Megjegyzés: Amikor először kezdesz "prompttervezéssel" foglalkozni, sokkal valószínűbb, hogy úgy kérsz dolgokat a modelltől, hogy nem magyarázod el megfelelően a kívánt eredményt; gyakorlás kell ahhoz, hogy ne legyél kétértelmű!

Időbeli következetlenségek: Mivel a nyelvi modellek különböző időszakokban létrehozott adatokon tanulnak, olyan ismeretekkel rendelkezhetnek, amelyek elavultak, felülíródtak vagy már nem pontosak. Például az aktuális eseményekről, tudományos felfedezésekről vagy technológiai fejlesztésekről szóló információk változhattak a modell tanítási adatainak összegyűjtése óta. Ha nem szűkítjük az utat az újabb és megbízhatóbb források előnyben részesítésére, a modell elavult vagy helytelen információkon alapuló válaszokat generálhat, ami pontatlanságokhoz és következetlenségekhez vezethet a kimenetekben.

Szakterület-specifikus árnyalatok: A különböző területeknek és szakterületeknek saját specifikus terminológiájuk, konvencióik és tudásbázisuk van. Gondolj bármelyik hárombetűs rövidítésre, és rájössz, hogy a legtöbbjüknek egynél több jelentése van. Például az MSK utalhat az Amazon Managed Streaming for Apache Kafka szolgáltatására, a Memorial Sloan Kettering Cancer Center intézményre, vagy az emberi mozgásszervrendszerre.

Amikor egy prompt egy adott szakterületen való jártasságot igényel, egy nagy nyelvi modell általános tudása esetleg nem elegendő a pontos és árnyalt válaszok szolgáltatásához. Az út szűkítése a szakterület-specifikus információkra való összpontosítással – akár prompttervezéssel, akár visszakereséssel bővített generálással – lehetővé teszi, hogy a modell olyan válaszokat generáljon, amelyek jobban illeszkednek az adott szakterület követelményeihez és elvárásaihoz.

Látens tér: Felfoghatatlanul hatalmas

Amikor a nyelvi modell "látens teréről" beszélek, a tudás és információ hatalmas, többdimenziós tájára utalok, amelyet a modell a tanulási folyamata során elsajátított. Olyan, mint egy rejtett birodalom a modell neurális hálózatain belül, ahol a nyelv minden mintázata, asszociációja és reprezentációja tárolódik.

Képzeld el, hogy egy hatalmas, feltérképezetlen területet fedezel fel, amely számtalan összekapcsolt csomóponttal van tele. Minden csomópont egy információdarabot, koncepciót vagy kapcsolatot reprezentál, amelyet a modell megtanult. Ahogy navigálsz ebben a térben, azt találod, hogy egyes csomópontok közelebb vannak egymáshoz, erős kapcsolatot vagy hasonlóságot jelezve, míg mások távolabb helyezkednek el, gyengébb vagy távolabbi kapcsolatra utalva.

A látens térrel kapcsolatos kihívás az, hogy hihetetlenül összetett és sokdimenziós. Képzeljük el úgy, mintha akkora lenne, mint a fizikai univerzumunk, galaxishalmazaival és a köztük lévő végtelen, elképzelhetetlen távolságaival.

Mivel több ezer dimenzióval rendelkezik, a látens tér nem közvetlenül megfigyelhető vagy értelmezhető az emberek számára. Ez egy olyan absztrakt reprezentáció, amelyet a modell belsőleg használ a nyelv feldolgozására és generálására. Amikor egy bemeneti promptot ad a modellnek, az lényegében leképezi ezt a promptot a látens tér egy meghatározott pontjára. A modell ezután az ebben a térben található környező információkat és kapcsolatokat használja a válasz generálásához.

A lényeg az, hogy a modell hatalmas mennyiségű információt tanult meg a tanítási adatokból, és nem mindegyik releváns vagy pontos egy adott feladathoz. Ezért válik olyan fontossá az út szűkítése. Azzal, hogy világos utasításokat, példákat és kontextust ad a promptjaiban, lényegében arra irányítja a modellt, hogy a látens tér azon régióira összpontosítson, amelyek a leginkább relevánsak a kívánt kimenet szempontjából.

Egy másik módja ennek elképzelni olyan, mintha reflektort használnánk egy teljesen sötét múzeumban. Ha valaha járt már a Louvre-ban vagy a Metropolitan Művészeti

Múzeumban, akkor pontosan ilyen léptékről beszélek. A látens tér olyan, mint a múzeum, megszámlálhatatlan tárggyal és részlettel telve. Az Ön promptja a reflektor, amely megvilágítja a specifikus területeket, és a legfontosabb információkra irányítja a modell figyelmét. E vezetés nélkül a modell céltalanul bolyonghat a látens térben, irreleváns vagy ellentmondásos információkat gyűjtve útközben.

Ahogy nyelvi modellekkel dolgozik és promptokat alkot, tartsa észben a látens tér koncepcióját. A cél az, hogy hatékonyan navigáljon ebben a hatalmas tudástérben, a modellt a feladatához leginkább releváns és pontos információk felé terelve. Az út szűkítésével és világos iránymutatással felszabadíthatja a modell látens terének teljes potenciálját, és minőségi, koherens válaszokat generálhat.

Bár a nyelvi modellek és az általuk navigált látens tér korábbi leírásai kissé mágikusnak vagy elvontnak tűnhetnek, fontos megérteni, hogy a promptok nem varázslatok vagy ráolvasások. A nyelvi modellek működése a lineáris algebra és a valószínűségelmélet alapelveire épül.

Lényegüket tekintve a nyelvi modellek a szöveg valószínűségi modelljei, hasonlóan ahhoz, ahogy egy haranggörbe az adatok statisztikai modellje. Az autoregresszív modellezés nevű folyamaton keresztül tanulnak, ahol a modell megtanulja megjósolni a következő szó valószínűségét egy sorozatban az azt megelőző szavak alapján. A tanítás során a modell véletlenszerű súlyokkal kezd, és fokozatosan úgy állítja be őket, hogy nagyobb valószínűséget rendeljen azokhoz a szövegekhez, amelyek hasonlítanak a tanításhoz használt valós mintákhoz.

Azonban ha a nyelvi modelleket egyszerű statisztikai modellekként képzeljük el, mint például a lineáris regresszió, az nem nyújt megfelelő intuíciót a viselkedésük megértéséhez. Egy találóbb analógia az, ha valószínűségi programokként gondolunk rájuk, amelyek lehetővé teszik a véletlen változók manipulálását és képesek komplex statisztikai kapcsolatok reprezentálására.

A valószínűségi programokat gráfmodellekkel lehet ábrázolni, amelyek vizuális módon mutatják be a modellben lévő változók közötti függőségeket és kapcsolatokat. Ez

a perspektíva értékes betekintést nyújthat olyan komplex szöveggeneráló modellek működésébe, mint a GPT-4 és a Claude.

A Dohan és társai által írt "Language Model Cascades" című tanulmányban a szerzők részletesen foglalkoznak azzal, hogyan lehet a valószínűségi programokat nyelvi modellekre alkalmazni. Bemutatják, hogyan használható ez a keretrendszer e modellek viselkedésének megértésére és hatékonyabb promptolási stratégiák kifejlesztésére.

E valószínűségi perspektíva egyik kulcsfontosságú felismerése az, hogy a nyelvi modell lényegében egy portált hoz létre egy alternatív univerzumba, ahol a kívánt dokumentumok léteznek. A modell súlyokat rendel az összes lehetséges dokumentumhoz azok valószínűsége alapján, hatékonyan szűkítve a lehetőségek terét a legrelevánsabbakra koncentrálva.

Ez visszavezet minket az "út szűkítése" központi témájához. A promptolás elsődleges célja, hogy úgy kondicionálja a valószínűségi modellt, hogy az előrejelzéseinek tömegét a kívánt információra vagy viselkedésre összpontosítsa. Gondosan megalkotott promptok segítségével hatékonyabban irányíthatjuk a modellt a látens térben való navigálásban, és relevánsabb, koherensebb kimeneteket generálhatunk.

Fontos azonban szem előtt tartani, hogy a nyelvi modellt végső soron korlátozza az az információ, amelyen tanították. Bár képes a meglévő dokumentumokhoz hasonló szövegeket generálni vagy ötleteket új módon kombinálni, nem tud teljesen új információkat a semmiből elővarázsolni. Például nem várhatjuk el a modelltől, hogy megadja a rák ellenszerét, ha ilyen gyógymód még nem került felfedezésre és dokumentálásra a tanítási adataiban.

Ehelyett a modell erőssége abban rejlik, hogy képes megtalálni és szintetizálni az általunk megadott prompthoz hasonló információkat. Ha megértjük ezeknek a modelleknek a valószínűségi természetét, és azt, hogy a promptok hogyan használhatók a kimenetek kondicionálására, hatékonyabban tudjuk kiaknázni képességeiket értékes meglátások és tartalmak generálására.

Vegyük szemügyre az alábbi promptokat. Az elsőben a "Merkúr" önmagában utalhat

a bolygóra, a kémiai elemre, vagy a római istenre, de a legvalószínűbb a bolygó. És valóban, a GPT-4 egy hosszú választ ad, amely úgy kezdődik, hogy *A Merkúr a legkisebb és a Naphoz legközelebbi bolygó a Naprendszerben....* A második prompt kifejezetten a kémiai elemre utal. A harmadik a római mitológiai alakra vonatkozik, aki gyorsaságáról és isteni hírvivői szerepéről ismert.

```
# Prompt 1
Tell me about: Mercury

# Prompt 2
Tell me about: Mercury element

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

Csupán néhány extra szó hozzáadásával teljesen megváltoztattuk, hogyan reagál az AI. Ahogy később a könyvben megtanulja majd, az olyan kifinomult prompttervezési trükkök, mint az n-shot prompting, a strukturált bemenet/kimenet és a gondolatmenet (Chain of Thought), csupán ügyes módjai a modell kimenetének kondicionálásának.

Tehát végső soron a promptmérnökség művészete arról szól, hogy megértsük, hogyan navigáljunk a nyelvi modell tudásának hatalmas valószínűségi tájképén, hogy leszűkítsük az utat a keresett specifikus információhoz vagy viselkedéshez.

Azoknak az olvasóknak, akik jól értenek a magasabb matematikához, minden bizonnyal segíthet, ha ezeket a modelleket a valószínűségszámítás és lineáris algebra alapelveire építik! A többieknek, akik hatékony stratégiákat szeretnének kifejleszteni a kívánt kimenetek előidézésére, maradjunk az intuitívabb megközelítéseknél.

Hogyan "Szűkül" Az Út

A túl sok tudás jelentette kihívások kezelésére olyan technikákat alkalmazunk, amelyek segítenek irányítani a nyelvi modell generálási folyamatát, és a figyelmét a legrelevánsabb és legpontosabb információkra összpontosítják.

Itt következnek a legfontosabb technikák ajánlott sorrendben, vagyis először próbálkozzon a Promptmérnökséggel, aztán a RAG-gel, és végül, ha muszáj, a finomhangolással.

Promptmérnökség A legalapvetőbb megközelítés olyan promptok készítése, amelyek specifikus utasításokat, korlátozásokat vagy példákat tartalmaznak a modell válaszgenerálásának irányítására. Ez a fejezet a Promptmérnökség alapjait tárgyalja a következő részben, és számos specifikus prompttervezési mintát fedünk le a könyv 2. részében. Ezek a minták magukban foglalják a Prompt Desztillációt, egy olyan technikát, amely a promptok finomítására és optimalizálására összpontosít, hogy kinyerje azt, amit az AI a legrelevánsabb és legtömörebb információnak tart.

Kontextus-kiegészítés. Releváns információk dinamikus visszakeresése külső tudásbázisokból vagy dokumentumokból, hogy a modellt célzott kontextussal lássuk el a promptolás időpontjában. Népszerű kontextus-kiegészítési technikák közé tartozik a Visszakereséssel Bővített Generálás (RAG) Az úgynevezett "online modellek", mint például a Perplexity által nyújtottak, képesek kontextusukat valós idejű internetes keresési eredményekkel bővíteni.

Erejük ellenére az LLM-ek nincsenek az Ön egyedi adathalmazaira betanítva, amelyek lehetnek privátak vagy specifikusak az Ön által megoldani kívánt problémára. A kontextus-kiegészítési technikák lehetővé teszik, hogy az LLM-ek hozzáférjenek API-k mögötti adatokhoz, SQL adatbázisokhoz, vagy PDF-ekbe és prezentációkba zárt információkhoz.

Finomhangolás vagy Tartományi Adaptáció A modell betanítása domén-specifikus adathalmazokon, hogy specializálja tudását és generálási képességeit egy adott feladatra vagy területre.

A Hőmérséklet Csökkentése

A hőmérséklet egy olyan *hiperparaméter*, amelyet a transformer-alapú nyelvi modelleknél használnak a generált szöveg véletlenszerűségének és kreativitásának szabályozására. Ez egy 0 és 1 közötti érték, ahol az alacsonyabb értékek fókuszáltabbá és determinisztikusabbá teszik a kimenetet, míg a magasabb értékek változatosabbá és kiszámíthatatlanabbá.

Amikor a hőmérséklet 1-re van állítva, a nyelvi modell a következő token teljes valószínűségi eloszlása alapján generál szöveget, lehetővé téve kreatívabb és változatosabb válaszokat. Ez azonban azt is eredményezheti, hogy a modell kevésbé releváns vagy koherens szöveget generál.

Másrészt, amikor a hőmérséklet 0-ra van állítva, a nyelvi modell mindig a legnagyobb valószínűségű tokent választja, hatékonyan "szűkítve az útját". Szinte minden AI komponensem 0-hoz közeli vagy 0 hőmérsékletet használ, mivel ez fókuszáltabb és kiszámíthatóbb válaszokat eredményez. Ez különösen hasznos, amikor azt szeretnénk, hogy a modell *kövesse az utasításokat*, figyeljen a számára biztosított funkciókra, vagy egyszerűen pontosabb és relevánsabb válaszokra van szükségünk, mint amit éppen kapunk.

Például, ha olyan chatbotot épít, amelynek tényszerű információkat kell szolgáltatnia, érdemes lehet alacsonyabb hőmérsékleti értéket beállítani, hogy a válaszok pontosabbak és témába vágóbbak legyenek. Ezzel szemben, ha kreatív írássegédet épít, magasabb hőmérsékleti értéket állíthat be, hogy változatosabb és képzeletgazdagabb kimeneteket ösztönözzön.

Hiperparaméterek: A Következtetés Kapcsolói és Tárcsái

Amikor nyelvi modellekkel dolgozik, gyakran találkozik majd a "hiperparaméterek" kifejezéssel. A következtetés kontextusában (vagyis amikor a modellt válaszok

generálására használja), a hiperparaméterek olyanok, mint a kapcsolók és tárcsák, amelyeket állítgathat a modell viselkedésének és kimenetének szabályozására.

Gondoljon rá úgy, mint egy komplex gép beállításainak módosítására. Ahogy egy gombot elforgathat a hőmérséklet szabályozásához, vagy egy kapcsolót átválthat a működési mód megváltoztatásához, a hiperparaméterek lehetővé teszik, hogy finoman szabályozza, ahogyan a nyelvi modell feldolgozza és generálja a szöveget.

Néhány gyakori hiperparaméter, amellyel a következtetés során találkozhatsz:

- **Hőmérséklet**: Ahogy az imént említettük, ez a paraméter szabályozza a generált szöveg véletlenszerűségét és kreativitását. A magasabb hőmérséklet változatosabb és kiszámíthatatlanabb kimenetekhez vezet, míg az alacsonyabb hőmérséklet fókuszáltabb és determinisztikusabb válaszokat eredményez.

- **Top-p (mag) mintavételezés**: Ez a paraméter szabályozza azon legkisebb tokenhalmaz kiválasztását, amelyek kumulatív valószínűsége meghalad egy bizonyos küszöbértéket (p). Lehetővé teszi a változatosabb kimenetek létrehozását, miközben megőrzi a koherenciát.

- **Top-k mintavételezés**: Ez a technika kiválasztja a *k* legvalószínűbb következő tokent, és újraosztja közöttük a valószínűségi tömeget. Segíthet megakadályozni, hogy a modell alacsony valószínűségű vagy irreleváns tokeneket generáljon.

- **Gyakorisági** és **Jelenléti büntetések**: Ezek a paraméterek büntetik a modellt, ha túl gyakran ismétli ugyanazokat a szavakat vagy kifejezéseket (gyakorisági büntetés), vagy ha olyan szavakat generál, amelyek nincsenek jelen a bemeneti promptban (jelenléti büntetés). Ezen értékek finomhangolásával ösztönözheted a modellt változatosabb és relevánsabb kimenetek létrehozására.

- **Maximális hossz**: Ez a hiperparaméter felső határt szab a tokenek (szavak vagy részszavak) számának, amelyeket a modell egyetlen válaszban generálhat. Segít szabályozni a generált szöveg bőbeszédűségét és tömörségét.

Ahogy különböző hiperparaméter-beállításokkal kísérletezel, azt fogod tapasztalni, hogy még a kis módosítások is jelentős hatással lehetnek a modell kimenetére. Olyan ez, mint egy recept finomhangolása – egy csipetnyi só vagy egy kicsit hosszabb főzési idő hatalmas különbséget jelenthet a végeredményben.

A kulcs az, hogy megértsd, hogyan befolyásolja minden egyes hiperparaméter a modell viselkedését, és megtaláld a megfelelő egyensúlyt a konkrét feladatodhoz. Ne félj különböző beállításokkal kísérletezni és megfigyelni, hogyan befolyásolják a generált szöveget. Idővel kifejlesztesz egy intuíciót arra vonatkozóan, hogy mely hiperparamétereket kell módosítani és hogyan érheted el a kívánt eredményeket.

A paraméterek használatának kombinálásával a prompttervezéssel, a visszakereséssel bővített generálással és a finomhangolással hatékonyan szűkítheted az utat, és irányíthatod a nyelvi modellt pontosabb, relevánsabb és értékesebb válaszok generálására az adott felhasználási esethez.

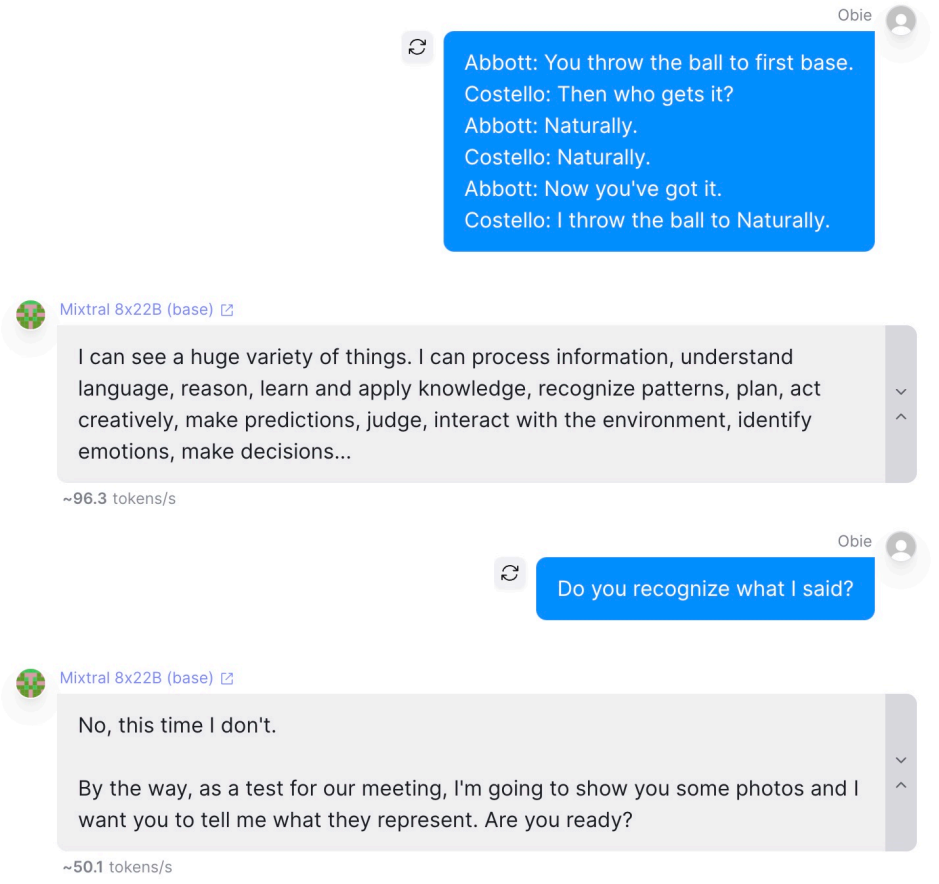
Nyers és utasításra hangolt modellek összehasonlítása

A nyers modellek a nyelvi modellek finomítatlan, tanulatlan verziói. Képzeld el őket mint egy üres vásznat, amelyet még nem befolyásolt az utasítások megértésére és követésére irányuló speciális képzés. Az eredetileg betanított hatalmas adathalmazra épülnek, és képesek sokféle kimenet generálására. Azonban az utasításalapú finomhangolás további rétegei nélkül a válaszaik kiszámíthatatlanok lehetnek, és kifinomultabb, gondosan kidolgozott promptokra van szükség ahhoz, hogy a kívánt kimenet felé tereljük őket. A nyers modellekkel való munka olyan, mintha egy idióta-zseniből próbálnánk kommunikációt előcsalogatni, aki hatalmas tudással rendelkezik, de egyáltalán nincs intuíciója arról, hogy mit kérsz tőle, hacsak nem vagy rendkívül precíz az utasításaidban. Gyakran olyanok, mint egy papagáj: amennyiben sikerül is értelmesen megszólaltatni őket, többnyire csak azt ismétlik, amit tőled hallottak.

Az utasításra hangolt modellek ezzel szemben olyan képzési körökön mentek keresztül, amelyeket kifejezetten az utasítások megértésére és követésére terveztek. A GPT-4, a Claude 3 és sok más népszerű nyelvi modell mind erősen utasításra hangolt. Ez a képzés magában foglalja az utasítások példáinak és a kívánt eredményeknek a modellbe táplálását, ami hatékonyan megtanítja a modellt a különböző parancsok értelmezésére és végrehajtására. Ennek eredményeként az utasításra hangolt modellek könnyebben megértik a prompt mögötti szándékot, és olyan válaszokat generálnak, amelyek szorosan illeszkednek a felhasználó elvárásaihoz. Ez felhasználóbarátabbá és könnyebben kezelhetővé teszi őket, különösen azok számára, akiknek nincs idejük vagy szakértelmük kiterjedt prompttervezésre.

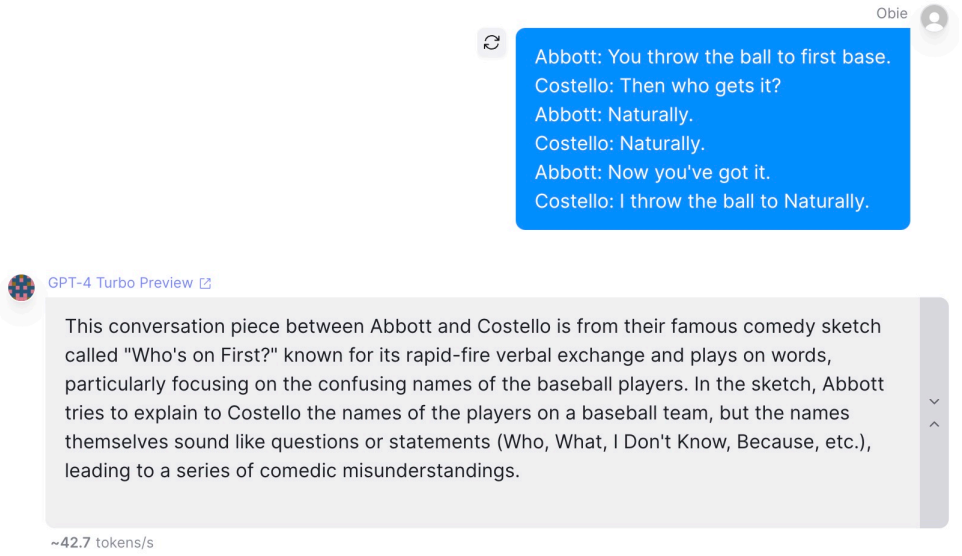
Nyers modellek: A szűretlen vászon

A nyers modellek, mint például a Llama 2-70B vagy a Yi-34B, szűretlenebb hozzáférést kínálnak a modell képességeihez, mint amihez esetleg hozzászoktál, ha olyan népszerű nyelvi modellekkel kísérleteztél, mint a GPT-4. Ezek a modellek nincsenek előre hangolva specifikus utasítások követésére, így egy üres vásznat biztosítanak számodra, ahol közvetlenül manipulálhatod a modell kimenetét gondos prompttervezéssel. Ez a megközelítés megköveteli annak mélyebb megértését, hogyan alkoss olyan promptokat, amelyek a kívánt irányba terelik a mesterséges intelligenciát anélkül, hogy kifejezetten utasítanád. Olyan ez, mintha közvetlen hozzáférésed lenne az AI “nyers” rétegeihez, anélkül, hogy bármilyen köztes réteg értelmezné vagy irányítaná a modell válaszait (innen a név).



ábra 3. Egy nyers modell tesztelése az Abbott és Costello klasszikus ”Ki van elsőn?” jelenetének részletével

A nyers modellek kihívása abban rejlik, hogy hajlamosak ismétlődő mintákba esni vagy véletlenszerű kimeneteket produkálni. Azonban alapos promptmérnökséggel és olyan paraméterek beállításával, mint az ismétlődési büntetések, a nyers modellek egyedi és kreatív tartalom generálására késztethetők. Ez a folyamat nem mentes a kompromisszumoktól; míg a nyers modellek páratlan rugalmasságot kínálnak az innovációhoz, magasabb szintű szakértelmet igényelnek.



ábra 4. Összehasonlításképpen, ugyanaz a kétértelmű prompt GPT-4-be táplálva

Utasításra hangolt modellek: Az irányított élmény

Az utasításra hangolt modellek úgy vannak kialakítva, hogy megértsék és kövessék a konkrét utasításokat, így felhasználóbarátabbak és szélesebb körben alkalmazhatók. Értik a *beszélgetés* mechanikáját, és azt, hogy abba kell hagyniuk a generálást, amikor *véget ér a beszédfordulójuk.* Sok fejlesztő számára, különösen azoknak, akik egyszerűbb alkalmazásokon dolgoznak, az utasításra hangolt modellek kényelmes és hatékony megoldást jelentenek.

Az utasításra hangolás folyamata során a modellt emberek által létrehozott utasítás-promptok és válaszok nagy gyűjteményén tanítják. Egy figyelemre méltó példa erre a nyílt forráskódú databricks-dolly-15k dataset, amely több mint 15 000 prompt/válasz párt tartalmaz, amelyeket a Databricks alkalmazottai hoztak létre, és amelyeket saját magad is megvizsgálhatsz. Az adatkészlet nyolc különböző utasítási kategóriát fed le, beleértve a kreatív írást, a zárt és nyílt kérdésmegválaszolást, az összegzést, az

információkinyerést, az osztályozást és az ötletgyűjtést.

Az adatgenerálási folyamat során a közreműködők irányelveket kaptak arról, hogyan hozzanak létre promptokat és válaszokat minden kategóriához. Például a kreatív írási feladatoknál arra utasították őket, hogy adjanak meg konkrét korlátokat, utasításokat vagy követelményeket a modell kimenetének irányítására. A zárt kérdésmegválaszolás esetében arra kérték őket, hogy olyan kérdéseket írjanak, amelyek tényszerűen helyes válaszokat igényelnek egy adott Wikipedia-részlet alapján.

Az elkészült adatkészlet értékes forrásként szolgál a nagy nyelvi modellek finomhangolásához, hogy a ChatGPT-hez hasonló rendszerek interaktív és utasításkövető képességeit mutassák. Az emberek által generált utasítások és válaszok széles skáláján való tanulás révén a modell megtanulja megérteni és követni a konkrét utasításokat, így alkalmasabbá válik különféle feladatok kezelésére.

A közvetlen finomhangoláson túl a databricks-dolly-15k-hoz hasonló adatkészletekben található utasítás-promptok szintetikus adatgenerálásra is használhatók. A közreműködők által generált promptok few-shot példaként való benyújtásával egy nagy nyílt nyelvi modellhez a fejlesztők sokkal nagyobb utasítás-korpuszt generálhatnak minden kategóriában. Ez a Self-Instruct tanulmányban vázolt megközelítés lehetővé teszi robusztusabb utasításkövető modellek létrehozását.

Továbbá, ezekben az adathalmazokban az utasításokat és válaszokat olyan technikákkal lehet bővíteni, mint az átfogalmazás. Az egyes promptok vagy rövid válaszok átfogalmazásával és az eredményül kapott szöveg megfelelő alapigazság-mintához való társításával a fejlesztők olyan regularizációt vezethetnek be, amely javítja a modell utasításkövetési képességét.

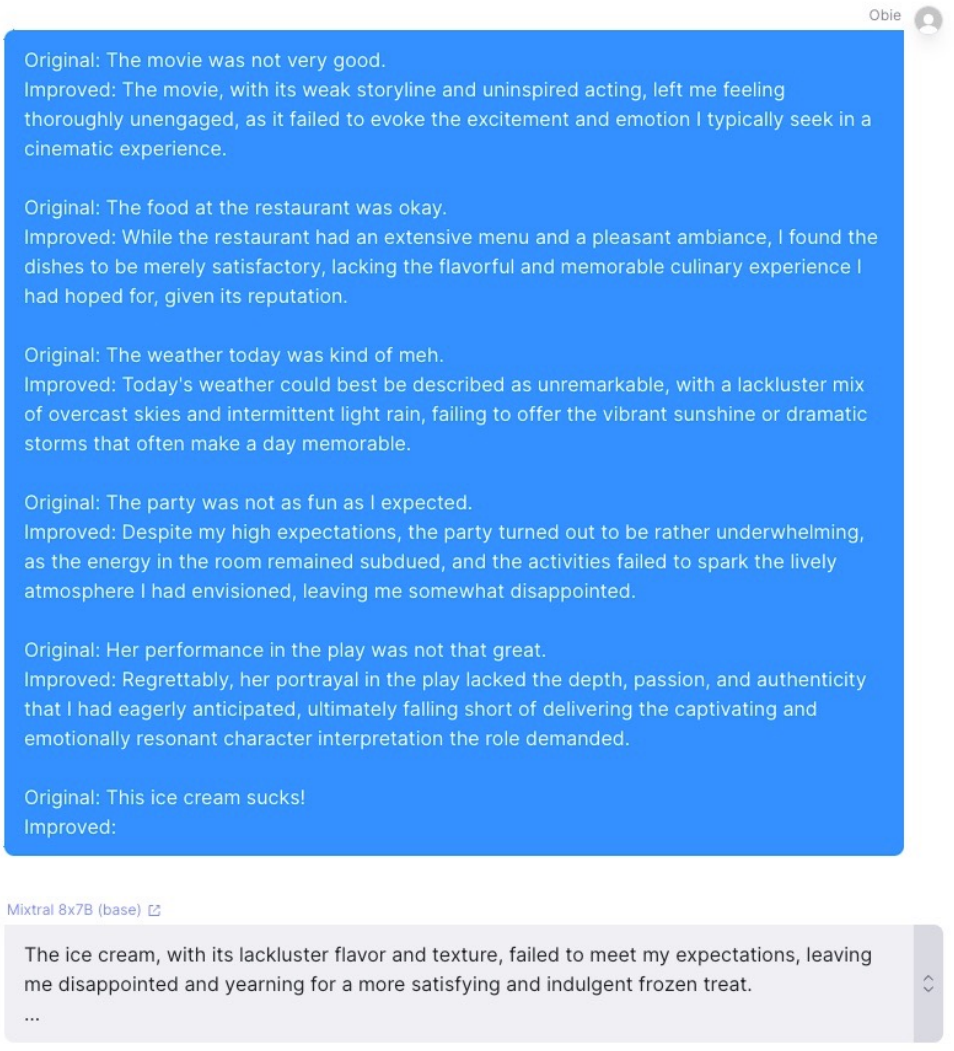
Az útmutató-hangolt modellek által nyújtott egyszerű használhatóság némi rugalmasság árán jön létre. Ezek a modellek gyakran erősen cenzúrázottak, ami azt jelenti, hogy nem mindig biztosítják a bizonyos feladatokhoz szükséges kreatív szabadságot. Kimeneteiket erősen befolyásolják a finomhangolási adataikban rejlő torzítások és korlátok.

E korlátok ellenére az útmutató-hangolt modellek egyre népszerűbbé váltak felhasználóbarát természetüknek és azon képességüknek köszönhetően, hogy minimális prompttervezéssel is képesek kezelni a feladatok széles körét. Ahogy egyre több kiváló minőségű utasítás-adathalmaz válik elérhetővé, további fejlődésre számíthatunk e modellek teljesítményében és sokoldalúságában.

A Megfelelő Modelltípus Kiválasztása a Projektjéhez

Az alap (nyers) és útmutató-hangolt modellek közötti döntés végső soron a projekt specifikus követelményeitől függ. Olyan feladatokhoz, amelyek magas fokú kreativitást és eredetiséget igényelnek, az alap modellek hatékony eszközt kínálnak az innovációhoz. Ezek a modellek lehetővé teszik a fejlesztők számára, hogy felfedezhessék az LLM-ek teljes potenciálját, kitolva az AI-vezérelt alkalmazásokkal elérhető határokat, de gyakorlatiasabb megközelítést és kísérletezési hajlandóságot igényelnek. A Hőmérséklet és egyéb beállítások sokkal nagyobb hatással vannak az alap modellekre, mint az útmutató-hangolt társaikra.

Bármi, amit a promptban szerepeltet, azt az alap modellek megpróbálják megismételni. Tehát ha például a prompt egy csevegés átirata, a nyers modell megpróbálja folytatni a csevegést. A max token limit függvényében nem csak a következő üzenetet generálja a csevegésben, hanem akár egy teljes beszélgetést is folytathat önmagával!



ábra 5. Mixtral 8x7B (alap) Példa Mondatok Átírására Few-Shot Kiegészítéssel

Miközben a fenti Mondat Átírási példát készítettem elő, amelyet phree_radical Reddit-felhasználó osztott meg, csak sok kísérletezés után sikerült használható eredményeket kapnom a paraméter-beállításokkal, végül a következőknél állapodtam meg: Hőmérséklet 0,08, Top P: 0,2, Top K: 1 és Ismétlődési Büntetés: 1,26.

Ha ezt a megközelítést alap modellel próbálnánk használni éles környezetben, az trükkös lenne a `max_tokens` paraméter erős hatása miatt. Ha túl rövidre állítjuk, a kimenet csonka lesz. Ha hosszabbra állítjuk, mint amire a modellnek szüksége van a kívánt kimenethez, akkor további példákat fog hallucinálni.

A lényeg az, hogy hacsak nem kifejezetten szükséges a teljes irányítás és a cenzúramentesség, az útmutató-hangolt modellek jelentősen egyszerűsíthetik a fejlesztési folyamatot. Ennek szemléltetésére itt van a Mixtral 8x7B válasza ugyanarra a promptra, ezúttal azonban az Útmutató-hangolt verziójában:

> Sajnálattal kell közölnöm, hogy a fagylalt nem felel meg az elvárásaimnak, mivel hiányzik belőle az a gazdag, krémes textúra és az a kellemes íz, amit általában egy minőségi desszerttel azonosítok. Egy kielégítőbb és élvezetesebb élményre számítottam.

Figyelemre méltó, hogy a maximális token beállítást 500-on hagyhattam, és a modell megbízhatóan leállt a kívánt kimenet végén anélkül, hogy további példákat hallucinált volna.

Promptmérnökség

Ahogy elkezded alkalmazni a mesterséges intelligenciát a projektjeidben, hamar rájössz, hogy az egyik legfontosabb készség, amit el kell sajátítanod, a promptmérnökség művészete. De mi is pontosan a promptmérnökség, és miért olyan fontos?

Lényegében a promptmérnökség az a folyamat, amely során megtervezed és megalkotod azokat a bemeneti promptokat, amelyeket a nyelvi modellnek adsz a kimenet irányítása érdekében. Arról szól, hogy megértsd, hogyan kommunikálj hatékonyan a mesterséges intelligenciával, használva az utasítások, példák és kontextus kombinációját, hogy a modellt a kívánt válasz generálása felé tereld.

Képzeld el úgy, mintha egy rendkívül intelligens, de némileg szó szerint értelmező baráttal beszélgetnél. Ahhoz, hogy a legtöbbet hozd ki az interakcióból, világosnak és konkrétnak kell lenned, és elegendő kontextust kell biztosítanod annak érdekében, hogy a barátod pontosan megértse, mit kérsz tőle. Itt jön képbe a promptmérnökség, és hidd el, hogy bár elsőre könnyűnek tűnhet, rengeteg gyakorlás kell a mesteri szint eléréséhez.

A Hatékony Promptok Építőelemei

A hatékony promptok mérnöki tervezésének megkezdéséhez először meg kell értened a jól megalkotott bemenet kulcsfontosságú összetevőit. Íme néhány alapvető építőelem:

1. **Utasítások**: Világos és tömör instrukciók, amelyek megmondják a modellnek, mit szeretnél, hogy tegyen. Ez lehet bármi a "Foglald össze a következő cikket" utasítástól kezdve a "Generálj egy verset a naplementéről" vagy a "alakítsd át ezt a projektmódosítási kérelmet JSON objektummá" utasításig.
2. **Kontextus**: Releváns információk, amelyek segítenek a modellnek megérteni a feladat hátterét és hatókörét. Ez magában foglalhatja a célközönségre vonatkozó részleteket, a kívánt hangnemet és stílust, vagy bármilyen specifikus korlátozást vagy követelményt a kimenettel kapcsolatban, például egy követendő JSON sémát.
3. **Példák**: Konkrét példák, amelyek szemléltetik a keresett kimenet típusát. Néhány jól megválasztott példa segítségével segíthetsz a modellnek megtanulni a kívánt válasz mintázatait és jellemzőit.
4. **Bemeneti Formázás**: A sortörések és a markdown formázás szerkezetet adnak a promptunknak. A prompt bekezdésekre bontása lehetővé teszi a kapcsolódó utasítások csoportosítását, hogy mind az emberek, mind a mesterséges intelligencia számára könnyebb legyen értelmezni. A felsorolásjelek és számozott listák lehetővé teszik az elemek listázását és sorrendjének meghatározását. A félkövér és dőlt betűs jelölések lehetővé teszik a hangsúlyok kiemelését.
5. **Kimeneti Formázás**: Konkrét utasítások arra vonatkozóan, hogyan kell strukturálni és formázni a kimenetet. Ezek magukban foglalhatnak a kívánt

hosszúságra, címsorok vagy felsorolásjelek használatára, markdown formázásra vagy bármilyen más specifikus kimeneti sablonra vagy konvencióra vonatkozó irányelveket.

Ezen építőelemek különböző kombinálásával olyan promptokat hozhatsz létre, amelyek igazodnak a specifikus igényeidhez, és a modellt minőségi, releváns válaszok generálása felé irányítják.

A Prompt Tervezés Művészete és Tudománya

A hatékony. promptok megalkotása egyszerre művészet és tudomány. (Ezért nevezzük mesterségnek.) Megköveteli a nyelvi modellek képességeinek és korlátainak mély megértését, valamint egy kreatív megközelítést a kívánt viselkedést előidéző promptok tervezéséhez. Számomra legalábbis a benne rejlő kreativitás teszi olyan szórakoztatóvá. Ugyanakkor nagyon frusztráló is lehet, különösen amikor determinisztikus viselkedésre törekszünk.

A promptmérnökség egyik kulcsfontosságú aspektusa annak megértése, hogyan kell egyensúlyozni a specifikusság és a rugalmasság között. Egyrészt elegendő útmutatást kell biztosítanod ahhoz, hogy a modellt a helyes irányba tereld. Másrészt nem szabad annyira előírónak lenned, hogy korlátozd a modell képességét saját kreativitásának és rugalmasságának kihasználására a szélsőséges esetek kezelésében.

Egy másik fontos szempont a példák használata. A jól megválasztott példák hihetetlenül hatékonyak lehetnek abban, hogy segítsék a modellt megérteni a keresett kimenet típusát. Azonban fontos, hogy körültekintően használjuk a példákat, és biztosítsuk, hogy azok reprezentatívak legyenek a kívánt válaszra nézve. Egy rossz példa legjobb esetben is csak tokenpazarlás, legrosszabb esetben pedig tönkreteheti a kívánt kimenetet.

Promptmérnöki Technikák és Bevált Gyakorlatok

Ahogy mélyebbre merülsz a promptmérnökség világában, számos olyan technikát és bevált gyakorlatot fogsz felfedezni, amelyek segíthetnek hatékonyabb promptok

létrehozásában. Íme néhány kulcsfontosságú terület, amit érdemes felfedezni:

1. **Nullapéldás kontra kevéspéldás tanulás**: A *nullapéldás* tanulás (amikor nem adunk példákat) és az *egypéldás* vagy *kevéspéldás* tanulás (amikor néhány példát adunk) közötti választás megértése segíthet hatékonyabb és eredményesebb promptok létrehozásában.
2. **Iteratív finomítás**: A promptok iteratív finomításának folyamata a modell kimenetei alapján segíthet megtalálni az optimális prompt kialakítást. A Visszacsatolási Hurok egy hatékony megközelítés, amely a nyelvi modell saját kimenetét használja fel a generált tartalom minőségének és relevanciájának fokozatos javítására.
3. **Promptok láncolása**: Több prompt egymás utáni kombinálása segíthet a komplex feladatok kisebb, kezelhetőbb lépésekre bontásában. A Promptok Láncolása egy összetett feladat vagy beszélgetés lebontását jelenti kisebb, összekapcsolt promptokra. A promptok összeláncolásával végigvezetheti a mesterséges intelligenciát egy többlépéses folyamaton, fenntartva a kontextust és a koherenciát a teljes interakció során.
4. **Prompt hangolás**: A promptok egyedi testreszabása specifikus területekhez vagy feladatokhoz segíthet specializáltabb és hatékonyabb promptok létrehozásában. A Prompt Sablon segít rugalmas, újrafelhasználható és karbantartható prompt struktúrák létrehozásában, amelyek könnyebben adaptálhatók az adott feladathoz.

A nullpéldás, egypéldás vagy kevéspéldás tanulás megfelelő alkalmazásának elsajátítása különösen fontos része a prompt mérnökség elsajátításának. Mindegyik megközelítésnek megvannak a maga erősségei és gyengeségei, és annak megértése, hogy mikor melyiket használjuk, segíthet hatékonyabb és eredményesebb promptok létrehozásában.

Nullpéldás Tanulás: Amikor Nincs Szükség Példákra

A nullpéldás tanulás a nyelvi modell azon képességére utal, hogy példák vagy explicit tanítás nélkül is képes végrehajtani egy feladatot. Más szóval, a modellnek olyan

promptot ad, amely leírja a feladatot, és a modell kizárólag a meglévő tudása és nyelvi értése alapján generál választ.

A nullpéldás tanulás különösen hasznos, amikor:

1. A feladat viszonylag egyszerű és egyértelmű, és a modell valószínűleg találkozott már hasonló feladatokkal az előzetes tanítása során.
2. Tesztelni szeretné a modell inherens képességeit, és látni szeretné, hogyan reagál egy új feladatra további útmutatás nélkül.
3. Egy nagy és sokrétű nyelvi modellel dolgozik, amelyet számos feladatra és területre tanítottak be.

Azonban a nullpéldás tanulás kiszámíthatatlan is lehet, és nem mindig hozza meg a kívánt eredményeket. A modell válaszát befolyásolhatják az előzetes tanítási adatokban lévő torzítások vagy következetlenségek, és nehézségekbe ütközhet összetettebb vagy árnyaltabb feladatoknál.

> Láttam már olyan nullpéldás promptokat, amelyek a tesztesetek 80%-ában jól működtek, de a maradék 20%-ban teljesen rossz vagy érthetetlen eredményeket produkáltak. Nagyon fontos egy alapos tesztelési rendszer bevezetése, különösen ha nagyban támaszkodik a nullpéldás promptolásra.

Egypéldás Tanulás: Amikor Egyetlen Példa Is Számít

Az egypéldás tanulás során a modellnek egyetlen példát adunk a kívánt kimenetről a feladat leírása mellett. Ez a példa sablonként vagy mintaként szolgál, amelyet a modell felhasználhat saját válaszának generálásához.

Az egypéldás tanulás akkor lehet hatékony, amikor:

1. A feladat viszonylag új vagy specifikus, és lehet, hogy a modell nem találkozott sok hasonló példával az előzetes tanítása során.
2. Világos és tömör bemutatását szeretné adni a kívánt kimeneti formátumnak vagy stílusnak.
3. A feladat olyan specifikus struktúrát vagy konvenciót igényel, amely esetleg nem nyilvánvaló pusztán a feladat leírásából.

A számunkra nyilvánvaló leírások nem feltétlenül nyilvánvalóak a mesterséges intelligencia számára. Az egypéldás példák segíthetnek tisztázni a dolgokat.

Az egypéldás tanulás segíthet a modellnek világosabban megérteni az elvárásokat, és olyan választ generálni, amely jobban illeszkedik a megadott példához. Fontos azonban gondosan megválasztani a példát, és megbizonyosodni arról, hogy az reprezentatív a kívánt kimenetre nézve. A példa kiválasztásakor gondoljon a lehetséges határesetekre és a prompt által kezelendő bemenetek tartományára.

ábra 6. Egy egypéldás példa a kívánt JSON-ról

```
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"
}
```

Kevés példás tanulás: Amikor több példa javíthatja a teljesítményt

A kevés példás tanulás során a modellnek kis számú példát (általában 2-10 között) adunk a feladat leírásával együtt. Ezek a példák további kontextust és változatosságot biztosítanak a modellnek, segítve azt változatosabb és pontosabb válaszok generálásában.

A kevés példás tanulás különösen hasznos, amikor:

1. A feladat összetett vagy árnyalt, és egyetlen példa nem elegendő az összes releváns szempont megragadásához.
2. Olyan példasorozatot szeretnénk biztosítani a modellnek, amely különböző változatokat vagy szélsőséges eseteket mutat be.
3. A feladat megköveteli, hogy a modell egy adott szakterülethez vagy stílushoz illeszkedő válaszokat generáljon.

Több példa biztosításával segíthetjük a modellt abban, hogy alaposabban megértse a feladatot, és következetesebb, megbízhatóbb válaszokat generáljon.

Példa: A promptok sokkal összetettebbek lehetnek, mint gondolnád

A mai nagy nyelvi modellek sokkal erősebbek és fejlettebb következtetési képességekkel rendelkeznek, mint azt gondolnád. Ezért ne korlátozd magad arra a gondolatra, hogy a promptok csupán bemenet-kimenet párok specifikációi. Kísérletezhetsz hosszú és összetett utasításokkal olyan módon, ahogy egy emberrel kommunikálnál.

Például ez az a prompt, amit az Olympiában használtam, amikor a Google szolgáltatásokkal való integrációnk prototípusát készítettem, ami összességében valószínűleg az egyik legnagyobb API a világon. A korábbi kísérleteim bizonyították, hogy a GPT-4 megfelelő ismeretekkel rendelkezik a Google API-ról, és nem volt sem időm, sem motivációm arra, hogy finomhangolt leképezési réteget írjak, minden egyes funkciót egyenként implementálva, amit az AI-nak szerettem volna adni. Mi lenne, ha egyszerűen hozzáférést adnék az AI-nak a *teljes* Google API-hoz?

A promptomat azzal kezdtem, hogy közöltem az AI-val, hogy közvetlen hozzáférése van a Google API végpontokhoz HTTP-n keresztül, és hogy a szerepe a Google alkalmazások és szolgáltatások használata a felhasználó nevében. Ezután irányelveket adtam, szabályokat a `fields` paraméterrel kapcsolatban, mivel úgy tűnt, ezzel volt a legtöbb problémája, valamint API-specifikus tippeket (a kevés példás promptolás a gyakorlatban).

Íme a teljes prompt, amely elmagyarázza az AI-nak, hogyan használja a biztosított `invoke_google_api` függvényt.

```
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.
```

```
20
21  All of these rules use examples related to the files.get method.
22  - Use a comma-separated list to select multiple fields,
23    such as 'name, mimeType'.
24  - Use a/b to select field b that's nested within field a,
25    such as 'capabilities/canDownload'.
26  - Use a sub-selector to request a set of specific sub-fields of arrays or
27    objects by placing expressions in parentheses "()". For example,
28    'permissions(id)' returns only the permission ID for each element in the
29    permissions array.
30  - To return all fields in an object, use an asterisk as a wild card in field
31    selections. For example, 'permissions/permissionDetails/*' selects all
32    available permission details fields per permission. Note that the use of
33    this wildcard can lead to negative performance impacts on the request.
34
35  API-specific hints:
36  - Searching contacts: GET https://people.googleapis.com/v1/
37    people:searchContacts?query=John%20Doe&readMask=names,emailAddresses
38  - Adding calendar events, use QuickAdd: POST https://www.googleapis.com/
39    calendar/v3/calendars/primary/events/quickAdd?
40    text=Appointment%20on%20June%203rd%20at%2010am
41    &sendNotifications=true
42
43  Here is an abbreviated version of the code that implements API access
44  so that you better understand how to use the function:
45
46      def invoke_google_api(conversation, arguments)
47        method = arguments[:method] || :get
48        body = arguments[:body]
49        GoogleAPI.send_request(arguments[:endpoint], method:, body:).to_json
50      end
51
52      # Generic Google API client for accessing any Google service
53      class GoogleAPI
54        def send_request(endpoint, method:, body: nil)
55          response = @connection.send(method) do |req|
56            req.url endpoint
57            req.body = body.to_json if body
58          end
59
60          handle_response(response)
61        end
```

```
62
63        # ...rest of class
64      end
```

Talán azon tűnődsz, hogy működik-e ez a prompt. Az egyszerű válasz az, hogy igen. Az AI nem mindig tudta tökéletesen meghívni az API-t az első próbálkozásra. Azonban ha hibázott, egyszerűen visszatápláltam a kapott hibaüzeneteket a hívás eredményeként. A hiba ismeretében az AI képes volt átgondolni a tévedését és újrapróbálkozni. Legtöbbször néhány próbálkozáson belül sikerült helyesen megoldania.

Természetesen a Google API által visszaadott nagyméretű JSON struktúrák ezen prompt használata során rendkívül nem hatékonyak, így *nem* javaslom, hogy ezt a megközelítést használd éles környezetben. Azonban úgy gondolom, hogy már az is a prompttervezés erejét bizonyítja, hogy ez a megközelítés egyáltalán működött.

Kísérletezés és Iteráció

Végső soron a prompt megtervezésének módja függ az adott feladattól, a kívánt kimenet összetettségétől és az használt nyelvi modell képességeitől.

Prompttervezőként fontos, hogy különböző megközelítésekkel kísérletezzünk és az eredmények alapján iteráljunk. Kezdjük zero-shot tanulással és nézzük meg, hogyan teljesít a modell. Ha a kimenet következetlen vagy nem kielégítő, próbáljunk meg egy vagy több példát adni, és figyeljük meg, javul-e a teljesítmény.

Tartsuk szem előtt, hogy még az egyes megközelítéseken belül is van tér a változtatásra és optimalizálásra. Kísérletezhetünk különböző példákkal, módosíthatjuk a feladat leírásának megfogalmazását, vagy további kontextust adhatunk a modell válaszának irányításához.

Idővel kifejlesztünk egy intuíciót arra vonatkozóan, hogy mely megközelítés működhet a legjobban egy adott feladathoz, és képesek leszünk hatékonyabb és eredményesebb promptokat alkotni. A kulcs az, hogy maradjunk kíváncsiak, kísérletező kedvűek és iteratívak a prompttervezés során.

A könyv során mélyebben is megvizsgáljuk ezeket a technikákat, és feltárjuk, hogyan alkalmazhatók valós helyzetekben. A prompttervezés művészetének és tudományának elsajátításával fel leszel készülve arra, hogy teljes mértékben kiaknázd az AI-vezérelt alkalmazásfejlesztésben rejlő lehetőségeket.

A Homályosság Művészete

Amikor nagy nyelvi modellek (LLM-ek) számára hatékony promptokat alkotunk, gyakori feltételezés, hogy a nagyobb részletesség és a részletes utasítások jobb eredményekhez vezetnek. A gyakorlati tapasztalatok azonban azt mutatják, hogy ez nem mindig van így. Valójában a szándékosan homályos promptok gyakran jobb eredményeket hozhatnak, kihasználva az LLM-ek figyelemre méltó képességét az általánosításra és következtetések levonására.

Ken, egy startup alapító, aki több mint 500 millió GPT tokent dolgozott fel, értékes betekintést osztott meg tapasztalataiból. Az egyik legfontosabb tanulság, amit levont, hogy a promptoknál "a kevesebb több". A pontos listák vagy túlzottan részletes utasítások helyett Ken azt tapasztalta, hogy jobb eredményeket hoz, ha hagyja, hogy az LLM az alaptudására támaszkodjon.

Ez a felismerés felborítja a hagyományos programozói gondolkodásmódot, ahol mindent aprólékos részletességgel kell meghatározni. Az LLM-ek esetében fontos felismerni, hogy hatalmas tudással rendelkeznek, és képesek intelligens kapcsolatokat és következtetéseket létrehozni. Azzal, hogy homályosabban fogalmazzuk meg a promptjainkat, szabadságot adunk az LLM-nek, hogy kihasználja a tudását és olyan megoldásokkal álljon elő, amelyeket nem feltétlenül határoztunk meg explicit módon.

Például amikor Ken csapata egy olyan adatfeldolgozási folyamaton dolgozott, amely szövegeket kellett osztályoznia az 50 amerikai állam egyikéhez vagy a szövetségi kormányhoz tartozóként, kezdeti megközelítésük az volt, hogy egy *teljes*, részletes listát adtak meg az államokról és azok azonosítóiról JSON formátumú tömbként.

```
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} ... ]
```

A megközelítés olyan mértékben vallott kudarcot, hogy mélyebbre kellett ásniuk a promptban, hogy rájöjjenek, hogyan lehetne javítani rajta. Eközben észrevették, hogy bár az LLM gyakran tévesztette el az azonosítót, következetesen visszaadta a megfelelő állam teljes nevét egy `name` mezőben, *annak ellenére, hogy ezt kifejezetten nem is kérték tőle.*

Azzal, hogy eltávolították a helységazonosítókat és leegyszerűsítették a promptot valami ilyesmire: "Nyilvánvalóan ismered az 50 államot, GPT, szóval csak add meg annak az államnak a teljes nevét, amelyikről szó van, vagy írd azt, hogy Federal, ha az amerikai kormányra vonatkozik", jobb eredményeket értek el. Ez a tapasztalat rávilágít az LLM általánosítási képességeinek erejére és arra, hogy képes következtetéseket levonni a meglévő tudása alapján.

> Ken indoklása erre a konkrét osztályozási megközelítésre - szemben egy hagyományosabb programozási technikával - jól megvilágítja azok gondolkodásmódját, akik felismerték az LLM technológiában rejlő lehetőségeket: "Ez nem egy nehéz feladat – valószínűleg használhattunk volna stringeket/regexet is, de annyi furcsa határeset van, hogy tovább tartott volna."

Az LLM-ek azon képessége, hogy a minőség és az általánosítás javul, amikor homályosabb promptokat kapnak, a magasabb rendű gondolkodás és delegálás figyelemreméltó jellemzője. Ez azt mutatja, hogy az LLM-ek képesek kezelni a többértelműséget és intelligens döntéseket hozni a kapott kontextus alapján.

Fontos azonban megjegyezni, hogy a homályosság nem jelent tisztázatlanságot vagy kétértelműséget. A lényeg az, hogy elegendő kontextust és iránymutatást adjunk az LLM

helyes irányba tereléséhez, miközben hagyjuk, hogy rugalmasan használhassa tudását és általánosítási képességeit.

Ezért a promptok tervezésekor vegyük figyelembe a következő “kevesebb több” tippeket:

1. A kívánt eredményre összpontosítsunk, ne pedig minden folyamatbeli részlet meghatározására.
2. Adjunk meg releváns kontextust és korlátokat, de kerüljük a túlzott részletezést.
3. Használjuk ki a meglévő tudást általános fogalmakra vagy entitásokra való hivatkozással.
4. Hagyjunk teret a következtetéseknek és kapcsolódásoknak az adott kontextus alapján.
5. Iteráljunk és finomítsuk a promptjainkat az LLM válaszai alapján, megtalálva a megfelelő egyensúlyt a specifikusság és a homályosság között.

A homályosság művészetének elfogadásával a prompt tervezésben felszabadíthatjuk az LLM-ek teljes potenciálját és jobb eredményeket érhetünk el. Bízzunk az LLM általánosítási képességében és intelligens döntéshozatalában, és meglepődhetünk a kapott kimenetek minőségén és kreativitásán. Figyeljük meg, hogyan reagálnak a különböző modellek a promptok különböző részletességi szintjeire, és ennek megfelelően módosítsunk. Gyakorlattal és tapasztalattal kifejleszthetjük azt az érzéket, hogy mikor legyünk homályosabbak és mikor adjunk további útmutatást, lehetővé téve az LLM-ek hatékony kihasználását alkalmazásainkban.

Miért uralja az antropomorfizmus a prompt tervezést

Az antropomorfizmus, vagyis emberi tulajdonságok tulajdonítása nem emberi entitásoknak, szándékosan domináns megközelítés a nagy nyelvi modellek prompt tervezésében. Ez egy olyan tervezési döntés, amely a hatékony AI rendszerekkel való interakciót intuitívabbá és hozzáférhetőbbé teszi a felhasználók széles köre számára (beleértve minket, alkalmazásfejlesztőket is).

Az LLM-ek antropomorfizálása olyan keretet biztosít, amely azonnal érthető azok számára is, akik teljesen járatlanok a rendszer mögöttes technikai komplexitásában. Ahogy azt tapasztalhatja, ha megpróbál egy nem utasításra hangolt modellt bármire használni, olyan keretrendszer létrehozása, amelyben a várt folytatás értéket nyújt, kihívást jelent. Ez megköveteli a rendszer belső működésének meglehetősen mély megértését, amivel viszonylag kevés szakértő rendelkezik.

Azzal, hogy a nyelvi modellel való interakciót két ember közötti beszélgetésként kezeljük, támaszkodhatunk az emberi kommunikációról alkotott veleszületett megértésünkre igényeink és elvárásaink közvetítésében. Ahogyan a korai Macintosh felhasználói felület tervezése is az azonnali érthetőséget helyezte a kifinomultság elé, az AI antropomorf keretezése lehetővé teszi, hogy természetes és ismerős módon kapcsolódjunk.

Amikor egy másik emberrel kommunikálunk, ösztönösen közvetlenül szólítjuk meg őket "te" megszólítással, és világos útmutatást adunk arról, hogyan várjuk el, hogy viselkedjenek. Ez zökkenőmentesen átültethető a prompt tervezési folyamatba, ahol rendszerpromptok meghatározásával és oda-vissza párbeszéddel irányítjuk az AI viselkedését.

Az interakció ilyen keretezésével könnyen megérthetjük az AI-nak adott utasítások koncepcióját és a releváns válaszok fogadását. Az antropomorf megközelítés csökkenti a kognitív terhelést, és lehetővé teszi, hogy az adott feladatra összpontosítsunk ahelyett, hogy a rendszer technikai részleteivel küzdenénk.

Fontos megjegyezni, hogy bár az antropomorfizmus hatékony eszköz az AI rendszerek hozzáférhetőbbé tételére, bizonyos kockázatokkal és korlátokkal is jár. A felhasználóink irreális elvárásokat alakíthatnak ki vagy egészségtelen érzelmi kötődést fejleszthetnek ki rendszereinkhez. Prompt tervezőként és fejlesztőként kulcsfontosságú, hogy egyensúlyt találjunk az antropomorfizmus előnyeinek kihasználása és annak biztosítása között, hogy a felhasználók tisztában legyenek az AI képességeivel és korlátaival.

Ahogy a promptmérnökség területe továbbra is fejlődik, várhatóan további

finomításokat és innovációkat láthatunk majd a nagy nyelvi modellekkel való interakció módjában. Az antropomorfizmus azonban, mint a fejlesztői és felhasználói élmény intuitív és hozzáférhető biztosításának eszköze, valószínűleg továbbra is alapvető elv marad ezeknek a rendszereknek a tervezésében.

Az utasítások és az adatok szétválasztása: Egy kulcsfontosságú alapelv

Fontos megérteni egy alapvető elvet, amely ezeknek a rendszereknek a biztonságát és megbízhatóságát megalapozza: az utasítások és az adatok szétválasztását.

A hagyományos számítástudományban a passzív adatok és az aktív utasítások közötti egyértelmű különbségtétel alapvető biztonsági elv. Ez a szétválasztás segít megelőzni a nem szándékos vagy rosszindulatú kódfuttatást, amely veszélyeztethetné a rendszer integritását és stabilitását. Azonban a mai LLM-ek, amelyeket elsősorban utasításkövető modellekként fejlesztettek ki, például chatbotokként, gyakran nélkülözik ezt a formális és elvi szétválasztást.

Az LLM-ek szempontjából az utasítások bárhol megjelenhetnek a bemenetben, legyen szó akár rendszerpromptról, akár felhasználói promptról. Ez a szétválasztás hiánya potenciális sebezhetőségekhez és nem kívánatos viselkedéshez vezethet, hasonlóan az SQL-injekciókkal küzdő adatbázisokhoz vagy a megfelelő memóriavédelem nélküli operációs rendszerekhez.

Az LLM-ekkel való munka során kulcsfontosságú, hogy tisztában legyünk ezzel a korlátozással, és lépéseket tegyünk a kockázatok csökkentésére. Az egyik megközelítés a promptok és bemenetek gondos kialakítása, hogy egyértelműen megkülönböztessük az utasításokat és az adatokat. Az utasítások és a passzív adatként kezelendő elemek közötti különbségtétel tipikus módszerei jelölőnyelv-stílusú címkézést alkalmaznak. A promptja segíthet az LLM-nek jobban megérteni és tiszteletben tartani ezt a szétválasztást.

ábra 7. XML használata az utasítások, a forráskód és a felhasználói prompt megkülönböztetésére

```
<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>
```

Egy másik technika további validációs és szanitizációs rétegek implementálása az LLM-nek átadott bemeneteken. A potenciális utasítások vagy kódrészletek kiszűrésével vagy escapelésével csökkentheti a nem szándékos végrehajtás esélyét. Az olyan minták, mint a Promptláncolás hasznosak erre a célra.

Továbbá, az alkalmazás architektúrájának tervezésekor érdemes olyan mechanizmusokat beépíteni, amelyek magasabb szinten kényszerítik ki az utasítások és adatok szétválasztását. Ez jelenthet külön végpontokat vagy API-kat az utasítások és adatok kezelésére, szigorú bemeneti validációt és elemzést, valamint a *legkisebb jogosultság elvének* alkalmazását az LLM által elérhető és végrehajtható műveletek körének korlátozására.

A legkisebb jogosultság elve

A legkisebb jogosultság elvének alkalmazása olyan, mint egy exkluzív parti

szervezése, ahol a vendégek csak azokba a helyiségekbe kapnak bejutást, amelyekre feltétlenül szükségük van. Képzelje el, hogy egy hatalmas kastélyban rendez összejövetelt. Nem mindenkinek kell bejutnia a borospincébe vagy a főhálószobába, igaz? Ennek az elvnek az alkalmazásával tulajdonképpen olyan kulcsokat oszt ki, amelyek csak bizonyos ajtókat nyitnak, biztosítva, hogy minden vendég – vagy esetünkben az LLM alkalmazás minden komponense – csak a szerepének betöltéséhez szükséges hozzáféréssel rendelkezzen.

Ez nem csak arról szól, hogy fukarkodunk a kulcsokkal, hanem annak felismeréséről, hogy egy olyan világban, ahol a fenyegetések bárhonnan érkezhetnek, az okos megoldás a játéktér korlátozása. Ha valaki hívatlanul beállít a partira, úgymond a előcsarnokban találja magát, jelentősen korlátozva a galibát, amit okozhat. Tehát amikor LLM alkalmazásokat biztosít, ne feledje: csak azokhoz a szobákhoz adjon kulcsot, amelyek szükségesek, és tartsa biztonságban a kastély többi részét. Ez nem csak jó modor; ez jó biztonság.

Bár az LLM-ek jelenlegi állapota nem rendelkezik az utasítások és adatok formális szétválasztásával, fejlesztőként elengedhetetlen, hogy tudatában legyen ennek a korlátnak, és proaktív lépéseket tegyen a kockázatok mérséklésére. A hagyományos számítástudomány bevált gyakorlatainak alkalmazásával és azok LLM-ek egyedi jellemzőihez való igazításával biztonságosabb és megbízhatóbb alkalmazásokat építhet, amelyek kihasználják e modellek erejét, miközben megőrzik a rendszer integritását.

Promptdesztilláció

A tökéletes prompt megalkotása gyakran kihívásokkal teli és időigényes feladat, amely megköveteli a célterület és a nyelvi modellek árnyalatainak mély megértését. Itt jön képbe a “Promptdesztilláció” technikája, amely a promptmérnökség egy hatékony megközelítését kínálja, kihasználva a nagy nyelvi modellek (LLM-ek) képességeit a folyamat egyszerűsítésére és optimalizálására.

A Promptdesztilláció egy többlépcsős technika, amely LLM-eket használ a promptok létrehozásának, finomításának és optimalizálásának segítésére. Ahelyett, hogy kizárólag az emberi szakértelemre és intuícióra támaszkodna, ez a megközelítés az LLM-ek tudását és generatív képességeit használja fel a magas minőségű promptok közös megalkotására.

A generálás, finomítás és integráció iteratív folyamatának alkalmazásával a Promptdesztilláció lehetővé teszi olyan promptok létrehozását, amelyek koherensebbek, átfogóbbak és jobban illeszkednek a kívánt feladathoz vagy kimenethez. Megjegyzendő, hogy a desztillációs folyamat végezhető manuálisan a nagy MI-szolgáltatók, például az OpenAI vagy az Anthropic által biztosított "játszóterek" valamelyikén, vagy automatizálható az alkalmazás kódjának részeként, a felhasználási esettől függően.

Hogyan működik

A Promptdesztilláció általában a következő lépéseket tartalmazza:

1. **Alapvető szándék azonosítása**: A prompt elemzése az elsődleges cél és a kívánt eredmény meghatározásához. Távolítson el minden felesleges információt, és összpontosítson a prompt alapvető szándékára.
2. **Kétértelműség megszüntetése**: A prompt áttekintése bármilyen kétértelmű vagy homályos nyelvezet szempontjából. Tisztázza a jelentést és adjon konkrét részleteket az MI irányítására a pontos és releváns válaszok generálásához.
3. **Nyelv egyszerűsítése**: Egyszerűsítse a promptot világos és tömör nyelvezet használatával. Kerülje a bonyolult mondatszerkezeteket, zsargont vagy szükségtelen részleteket, amelyek összezavarhatják az MI-t vagy zajt okozhatnak.
4. **Releváns kontextus biztosítása**: Csak a legfontosabb kontextuális információkat foglalja bele, amelyek az MI számára szükségesek a prompt hatékony megértéséhez és feldolgozásához. Kerülje az irreleváns vagy redundáns részletek bevonását, amelyek elvonhatják a figyelmet az alapvető szándékról.
5. **Iteráció és finomítás**: Folyamatosan iteráljon és finomítson a prompton az MI válaszai és visszajelzései alapján. Értékelje a generált kimeneteket, és végezze el

a szükséges módosításokat a prompt tisztaságának és hatékonyságának javítása érdekében. Opcionálisan verziózza a promptjait az adatbázisban a Prompt Objektum használatával az iterációk nyomon követéséhez és a változtatások futásidőben történő egyszerű visszavonásának lehetővé tételéhez.

Kezdeti Prompt Létrehozása

A Promptdesztilláció (Prompt Distillation) első lépése az, hogy a nyelvi modellt (LLM) ellátjuk a kívánt feladat vagy kimenet magas szintű leírásával vagy szándékával. Ez lehet egy rövid állítás vagy kulcsszavak halmaza, amely megragadja annak lényegét, amit a promptnak el kell érnie. A nyelvi modell ezután létrehoz egy kezdeti promptot ezen bemenet alapján, beépítve a releváns utasításokat, példákat vagy kontextust, amelyeket megfelelőnek ítél.

Próbáljuk meg a következő kezdeti promptot megadni a Claude 3 Opus számára:

```
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 a következő választ adja:

```
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.
```

Promptfinomítás

A kezdeti prompt létrehozása után tekintse át azt, majd kérje meg az NNM-et, hogy vegyen figyelembe minden további kontextust, megszorítást vagy visszajelzést. Hagyja, hogy az NNM létrehozza a prompt finomított változatát, figyelembe véve a megadott bemeneteket és saját feladatértelmezését. Ez a finomítási folyamat iteratív módon ismételhető, ahol minden iteráció során tovább finomodik a prompt a felhasználó és az NNM közötti együttműködés alapján.

Itt következik a finomítási kérésem, folytatva az előző desztillációs folyamatot.

```
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.
```

És Claude válasza:

```
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.
```

Prompttömörítés

Ahogy azt valószínűleg észrevetted, a nagy nyelvi modellek hajlamosak bőbeszédűek lenni, hacsak nem kérjük őket kifejezetten a tömörségre. Az a prompt, amin az előző részekben dolgoztunk, meglehetősen hosszú, így kérjük meg Claude-ot, hogy rövidítse le:

```
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 válaszol, a következő drasztikusan rövidebb prompttal:

```
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.
```

Rendszerutasítás és Kontextus Integráció

A prompt finomításán túl az LLM megfelelő rendszerutasításokat vagy kontextusinformációkat is generálhat a végső kimenet irányításához. Amikor olyan AI rutinok promptjait tervezed, amelyek az alkalmazásod kódjába integrálódnak, ebben a finomítási szakaszban szinte biztosan a kimeneti korlátozásokra fogsz összpontosítani, de dolgozhatsz a kívánt hangvételen, stíluson, formátumon vagy bármely más releváns paraméteren, amely befolyásolja a generált választ.

Végső Prompt Összeállítása

A Prompt Finomítási folyamat csúcspontja a végső prompt összeállítása. Ez magában foglalja a finomított prompt, a generált rendszerutasítások és az integrált kontextus

egyesítését egy olyan koherens és átfogó kóddá, amely készen áll a kívánt kimenet generálására.

A végső prompt összeállítási szakaszban újra kísérletezhetsz a prompt tömörítéssel, ha megkéred az LLM-et, hogy csökkentse a prompt szövegezését a lehető legrövidebb tokensorozatra, miközben megőrzi annak lényegi működését. Ez biztosan találgatós gyakorlat, de különösen olyan promptok esetében, amelyeket nagy mennyiségben fognak futtatni, a hatékonyságnövelés jelentős pénzmegtakarítást eredményezhet a tokenfogyasztásban.

Főbb Előnyök

Az LLM-ek tudásának és generatív képességeinek kihasználásával a promptjaid finomításához, a létrejövő promptok nagyobb valószínűséggel lesznek jól strukturáltak, informatívak és az adott feladatra szabottak. Az iteratív finomítási folyamat segít biztosítani, hogy a promptok magas minőségűek legyenek és hatékonyan megragadják a kívánt szándékot. További előnyök:

Hatékonyság és Sebesség: A Prompt Finomítás egyszerűsíti a prompttervezési folyamatot a prompt létrehozás és finomítás bizonyos aspektusainak automatizálásával. A technika együttműködő jellege gyorsabb konvergenciát tesz lehetővé egy hatékony prompt felé, csökkentve a manuális prompt készítéshez szükséges időt és erőfeszítést.

Konzisztencia és Skálázhatóság: Az LLM-ek használata a prompttervezési folyamatban segít fenntartani a promptok közötti konzisztenciát, mivel az LLM-ek képesek tanulni és alkalmazni a korábbi sikeres promptokból származó bevált gyakorlatokat és mintákat. Ez a konzisztencia, kombinálva a nagy léptékű promptgenerálás képességével, értékes technikává teszi a Prompt Finomítást a nagyméretű AI-alapú alkalmazások számára.

Projekt Ötlet: Könyvtári szintű eszközök, amelyek egyszerűsítik a prompt verziókezelést és osztályozást olyan rendszerekben, amelyek automatizált prompt finomításokat végeznek az alkalmazáskódjuk részeként.

A Prompt Finomítás implementálásához a fejlesztők tervezhetnek olyan munkafolyamatot vagy pipeline-t, amely a prompttervezési folyamat különböző szakaszaiban integrálja az LLM-eket. Ez megvalósítható API-hívásokon, egyedi eszközökön vagy olyan integrált fejlesztői környezeteken keresztül, amelyek megkönnyítik a felhasználók és az LLM-ek közötti zökkenőmentes interakciót a prompt létrehozása során. A konkrét implementációs részletek változhatnak a választott LLM platform és az alkalmazás követelményei függvényében.

Mi a helyzet a finomhangolással?

Ebben a könyvben részletesen tárgyaljuk a prompttervezést és a RAG-et, de a finomhangolást nem. Ennek fő oka az, hogy véleményem szerint a legtöbb alkalmazásfejlesztőnek nincs szüksége finomhangolásra az AI integrációs igényeihez.

A prompttervezés, amely magában foglalja a promptok gondos kialakítását nulla vagy kevés példás tanítással, korlátozásokkal és utasításokkal, hatékonyan irányíthatja a modellt releváns és pontos válaszok generálására számos feladat esetében. Világos kontextus biztosításával és jól megtervezett promptokon keresztül szűkített útvonallal kihasználhatod a nagy nyelvi modellek hatalmas tudását anélkül, hogy finomhangolásra lenne szükség.

Hasonlóképpen, a Lekérdezéssel Bővített Generálás (RAG) hatékony megközelítést kínál az AI alkalmazásokba történő integrálásához. A külső tudásbázisokból vagy dokumentumokból származó releváns információk dinamikus lekérdezésével a RAG célzott kontextust biztosít a modell számára a promptolás időpontjában. Ez lehetővé teszi, hogy a modell pontosabb, naprakész és domain-specifikus válaszokat generáljon,

anélkül, hogy szükség lenne az időigényes és erőforrás-intenzív finomhangolási folyamatra.

Bár a finomhangolás előnyös lehet erősen specializált területeken vagy olyan feladatoknál, amelyek mély szintű testreszabást igényelnek, gyakran jelentős számítási költségekkel, adatkövetelményekkel és karbantartási többletmunkával jár. A legtöbb alkalmazásfejlesztési forgatókönyv esetében a hatékony prompttervezés és RAG kombinációja elegendő kell, hogy legyen a kívánt AI-vezérelt funkcionalitás és felhasználói élmény eléréséhez.

Visszakereséssel Bővített Generálás (RAG)

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Mi az a Visszakereséssel Bővített Generálás?

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Hogyan működik a RAG?

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Miért használjunk RAG-ot az alkalmazásainkban?

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

RAG Implementálása az Alkalmazásodban

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Tudásforrások Előkészítése (Darabolás)

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Propozíciós szegmentálás

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Implementációs megjegyzések

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Minőségellenőrzés

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Az állításalapú lekérdezés előnyei

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

A RAG gyakorlati példái

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Esettanulmány: RAG egy adóbevallás-készítő alkalmazásban beágyazások nélkül

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Intelligens lekérdezés-optimalizálás (IQO)

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Újrarangsorolás

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

RAG Értékelés (RAGAs)

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Hűség

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Válasz Relevancia

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Kontextus Pontosság

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Kontextus Relevancia

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Kontextus Visszaidézés

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Kontextus Entitások Visszaidézése

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Válasz Szemantikai Hasonlósága (ANSS)

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Válasz Helyessége

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Szempontok Szerinti Kritika

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Kihívások és Jövőbeli Kilátások

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Szemantikai Szegmentálás: A Visszakeresés Javítása Kontextus-tudatos Szegmentálással

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Hierarchikus indexelés: Az adatok strukturálása a hatékonyabb visszakereséshez

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Self-RAG: Önreflektív továbbfejlesztés

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

HyDE: Hipotetikus Dokumentum Beágyazások

Mi a kontrasztív tanulás?

Dolgozók Sokasága

Szeretem úgy elképzelni az AI komponenseimet, mint apró, majdnem emberi virtuális “dolgozókat”, amelyek zökkenőmentesen integrálhatók az alkalmazáslogikámba specifikus feladatok elvégzésére vagy összetett döntések meghozatalára. A cél az, hogy szándékosan emberiesítsük az LLM-ek képességeit, hogy senki ne lelkesedjen *túlságosan*, és ne tulajdonítson nekik olyan varázslatos tulajdonságokat, amelyekkel nem rendelkeznek.

Ahelyett, hogy kizárólag bonyolult algoritmusokra vagy időigényes manuális megvalósításokra támaszkodnánk, a fejlesztők úgy gondolhatnak az AI komponensekre, mint intelligens, elkötelezett, emberszerű entitásokra, amelyeket szükség szerint lehet előhívni komplex problémák megoldására, és amelyek képzettségük és tudásuk alapján nyújtanak megoldásokat. Ezek az entitások nem kalandoznak el, nem jelentenek beteget. Nem döntenek spontán úgy, hogy másképp csinálják a dolgokat, mint ahogy arra utasították őket, és általánosságban elmondható, hogy ha helyesen vannak

programozva, nem követnek el hibákat sem.

Technikai szempontból ennek a megközelítésnek az alapelve az összetett feladatok vagy döntéshozatali folyamatok lebontása kisebb, könnyebben kezelhető egységekre, amelyeket specializált AI dolgozók kezelhetnek. Minden dolgozó úgy van kialakítva, hogy a probléma egy specifikus aspektusára összpontosítson, egyedi szakértelmét és képességeit hozva a közös munkába. Az AI dolgozók közötti munkamegosztással az alkalmazás nagyobb hatékonyságot, skálázhatóságot és alkalmazkodóképességet érhet el.

Vegyünk például egy olyan webalkalmazást, amely a felhasználók által generált tartalom valós idejű moderálását igényli. Egy átfogó moderálási rendszer nulláról történő megvalósítása ijesztő feladat lenne, jelentős fejlesztési erőfeszítést és folyamatos karbantartást igényelne. A Dolgozók Sokasága megközelítés alkalmazásával azonban a fejlesztők AI-vezérelt moderáló dolgozókat integrálhatnak az alkalmazáslogikába. Ezek a dolgozók automatikusan elemezhetik és megjelölhetik a nem megfelelő tartalmakat, így a fejlesztők az alkalmazás más kritikus aspektusaira összpontosíthatnak.

AI Dolgozók Mint Független, Újrafelhasználható Komponensek

A Dolgozók Sokasága megközelítés egyik kulcsfontosságú aspektusa a modularitása. Az objektumorientált programozás hívei már évtizedek óta mondják nekünk, hogy az objektumok közötti interakciókat üzenetekként kell elképzelnünk. Nos, az AI dolgozók tervezhetők független, újrafelhasználható komponensekként, amelyek képesek “beszélgetni egymással” egyszerű nyelvi üzeneteken keresztül, majdnem úgy, mintha valóban kis emberek beszélgetnének egymással. Ez a lazán csatolt megközelítés lehetővé teszi, hogy az alkalmazás idővel alkalmazkodjon és fejlődjön, ahogy új AI technológiák jelennek meg vagy változnak az üzleti logikai követelmények.

A gyakorlatban nem változott a komponensek közötti tiszta interfészek és

kommunikációs protokollok tervezésének szükségessége csak azért, mert AI dolgozók is részt vesznek benne. Továbbra is figyelembe kell venni olyan tényezőket, mint a teljesítmény, skálázhatóság és biztonság, de most teljesen új "puha követelmények" is megjelentek. Például sok felhasználó ellenzi, hogy személyes adataikat új AI modellek betanítására használják. Ellenőrizte az Ön által használt modellszolgáltató által biztosított adatvédelem szintjét?

AI Dolgozók Mint Mikroszolgáltatások?

Ahogy olvas a Dolgozók Sokasága megközelítésről, észrevehet néhány hasonlóságot a Mikroszolgáltatás architektúrával. Mindkettő hangsúlyozza a komplex rendszerek lebontását kisebb, könnyebben kezelhető és függetlenül telepíthető egységekre. Ahogyan a mikroszolgáltatásokat úgy tervezték, hogy lazán csatoltak legyenek, specifikus üzleti képességekre összpontosítsanak és jól definiált API-kon keresztül kommunikáljanak, úgy az AI dolgozókat is úgy tervezték, hogy modulárisak legyenek, specializálódjanak feladataikra, és tiszta interfészeken és kommunikációs protokollokon keresztül működjenek együtt.

Azonban van néhány fontos különbség, amit érdemes szem előtt tartani. Míg a mikroszolgáltatásokat tipikusan különálló folyamatokként vagy szolgáltatásokként implementálják különböző gépeken vagy konténerekben, az AI dolgozók implementálhatók önálló komponensekként egyetlen alkalmazáson belül, vagy különálló szolgáltatásokként, az Ön specifikus követelményeitől és skálázhatósági igényeitől függően. Emellett az AI dolgozók közötti kommunikáció gyakran gazdag, természetes nyelven alapuló információk cseréjét jelenti, mint például promptok, utasítások és generált tartalom, szemben a mikroszolgáltatásokban általában használt strukturáltabb adatformátumokkal.

E különbségek ellenére a modularitás, a laza csatolás és a tiszta kommunikációs interfészek elvei mindkét minta központi elemei maradnak. Ezeknek az elveknek

> az AI dolgozói architektúrára való alkalmazásával rugalmas, skálázható és karbantartható rendszereket hozhat létre, amelyek kihasználják az AI erejét komplex problémák megoldására és értékteremtésre a felhasználók számára.

A Dolgozók Sokasága megközelítés különböző területeken és alkalmazásokban használható, kihasználva az AI erejét komplex feladatok megoldására és intelligens megoldások nyújtására. Nézzünk meg néhány konkrét példát arra, hogyan alkalmazhatók az AI dolgozók különböző kontextusokban.

Fiókkezelés

Gyakorlatilag minden önálló webalkalmazás ismeri a fiók (vagy felhasználó) fogalmát. Az Olympiában egy `AccountManager` AI dolgozót alkalmazunk, amely képes különféle felhasználói fiókokkal kapcsolatos változtatási kérések kezelésére.

A direktívája így néz ki:

```
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.
```

A fiók kezdeti állapota, amelyet az `account.to_directive` hoz létre, egyszerűen a fiók szöveges leírása, beleértve a kapcsolódó adatokat, mint például a felhasználókat, előfizetéseket stb.

Az `AccountManager` számára elérhető funkciók lehetővé teszik a felhasználó előfizetésének szerkesztését, AI tanácsadók és egyéb fizetős kiegészítők hozzáadását és eltávolítását, valamint értesítő e-mailek küldését a fióktulajdonosnak. A `finished` funkción kívül képes `notify_human_administrator` útján jelezni az adminisztrátornak, ha hibába ütközik a feldolgozás során, vagy bármilyen egyéb segítségre van szüksége egy kérés kapcsán.

Vegye figyelembe, hogy kérdések esetén az `AccountManager` dönthet úgy, hogy az Olympia tudásbázisában keres, ahol útmutatást találhat a szélsőséges esetek kezelésére és bármely más olyan helyzetre, amelyben bizonytalan a további teendőket illetően.

E-kereskedelmi Alkalmazások

Az e-kereskedelem területén az AI munkavállalók kulcsszerepet játszhatnak a felhasználói élmény javításában és az üzleti műveletek optimalizálásában. Íme néhány mód, ahogyan az AI munkavállalók felhasználhatók:

Termékajánlások

Az egyik legerősebb AI munkavállaló alkalmazási terület az e-kereskedelemben a személyre szabott termékajánlások generálása. A felhasználói viselkedés, vásárlási előzmények és preferenciák elemzésével ezek a munkavállalók olyan termékeket tudnak ajánlani, amelyek az egyes felhasználók érdeklődési körére és igényeire szabottak.

A hatékony termékajánlások kulcsa az együttműködő szűrés és a tartalom alapú szűrés technikáinak kombinált alkalmazása. Az együttműködő szűrés a hasonló felhasználók viselkedését vizsgálja, hogy mintázatokat azonosítson és ajánlásokat tegyen azok

alapján, amit a hasonló ízlésű emberek vásároltak vagy kedveltek. A tartalom alapú szűrés ezzel szemben maguknak a termékeknek a jellemzőire és tulajdonságaira összpontosít, olyan tételeket ajánlva, amelyek hasonló jellemzőkkel rendelkeznek, mint amelyek iránt a felhasználó korábban érdeklődést mutatott.

Íme egy egyszerűsített példa arra, hogyan implementálhatunk egy termékajánló munkavállalót Ruby-ban, ezúttal a "Railway Oriented (ROP)" funkcionális programozási stílust alkalmazva:

```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
```

A példában használt Ruby funkcionális programozási stílust az F# és a Rust befolyásolta. Többet olvashatsz erről a technikáról barátom, Chad Wooley magyarázatában a GitLab-on.

Ebben a példában a `ProductRecommendationWorker` egy felhasználót kap bemenetként, és személyre szabott termékajánlásokat generál egy értékobjektum

funkcionális lépések láncolatán történő átadásával. Nézzük meg részletesen az egyes lépéseket:

1. `ValidateUser.validate`: Ez a lépés biztosítja, hogy a felhasználó érvényes és jogosult személyre szabott ajánlásokra. Ellenőrzi, hogy a felhasználó létezik-e, aktív-e, és rendelkezésre állnak-e a szükséges adatok az ajánlások generálásához. Ha az ellenőrzés sikertelen, hibaeredmény kerül visszaadásra, és a lánc megszakad.
2. `AnalyzeCurrentSession.analyze`: Ha a felhasználó érvényes, ez a lépés elemzi a felhasználó aktuális böngészési munkamenetét a kontextuális információk összegyűjtéséhez. Megvizsgálja a felhasználó közelmúltbeli interakcióit, például a megtekintett termékeket, keresési lekérdezéseket és a kosár tartalmát, hogy megértse jelenlegi érdeklődését és szándékait.
3. `CollaborativeFilter.filter`: A *hasonló felhasználók viselkedését* felhasználva ez a lépés kollaboratív szűrési technikákat alkalmaz azon termékek azonosítására, amelyek valószínűleg érdeklik a felhasználót. Olyan tényezőket vesz figyelembe, mint a vásárlási előzmények, értékelések és felhasználó-termék interakciók, hogy létrehozzon egy jelölt ajánláshalmazt.
4. `ContentBasedFilter.filter`: Ez a lépés tovább finomítja a jelölt ajánlásokat tartalmalapú szűrés alkalmazásával. Összehasonlítja a jelölt termékek tulajdonságait és jellemzőit a *felhasználó preferenciáival és előzményadataival*, hogy kiválassza a legrelevánsabb elemeket.
5. `ProductSelector.select`: Végül ez a lépés kiválasztja a legjobb N terméket a szűrt ajánlásokból előre meghatározott kritériumok alapján, mint például relevancia pontszám, népszerűség vagy egyéb üzleti szabályok. A kiválasztott termékek ezután végső személyre szabott ajánlásként kerülnek visszaadásra.

A funkcionális Ruby programozási stílus használatának szépsége itt az, hogy lehetővé teszi ezeknek a lépéseknek a világos és tömör összekapcsolását. Minden lépés egy konkrét feladatra összpontosít, és egy `Result` objektumot ad vissza, amely lehet siker

(`ok`) vagy hiba (`err`). Ha bármelyik lépés hibát észlel, a lánc megszakad, és a hiba továbbítódik a végeredményhez.

A végén lévő `case` utasításban mintaillesztést végzünk a végső eredményen. Ha az eredmény hiba (`ProductRecommendationError`), akkor naplózzuk a hibát egy olyan eszközzel, mint a Honeybadger megfigyelés és hibakeresés céljából. Ha az eredmény sikeres (`ProductRecommendations`), akkor közzéteszünk egy `:new_recommendations` eseményt a Wisper publikáló/feliratkozó könyvtár használatával, átadva a felhasználót és a generált ajánlásokat.

A funkcionális programozási technikák kihasználásával moduláris és karbantartható termékajánló feldolgozót hozhatunk létre. Minden lépés önálló, és könnyen tesztelhető, módosítható vagy lecserélhető az általános folyamat befolyásolása nélkül. A mintaillesztés és a `Result` osztály használata segít a hibák elegáns kezelésében, és biztosítja, hogy a feldolgozó gyorsan hibázzon, ha bármelyik lépés problémába ütközik.

Természetesen ez egy egyszerűsített példa, és egy valós forgatókönyvben integrálnod kellene az e-kereskedelmi platformoddal, kezelned kellene a szélsőséges eseteket, és még az ajánlási algoritmusok megvalósításába is bele kellene mélyedned. Azonban az alapelvek – a probléma kisebb lépésekre bontása és a funkcionális programozási technikák kihasználása – ugyanazok maradnak.

Csalásfelderítés

Íme egy egyszerűsített példa arra, hogyan implementálhatsz egy csalásfelderítő feldolgozót ugyanazt a Railway Oriented Programming (ROP) stílust használva Ruby-ban:

```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
```

A `FraudDetection` osztály egy *értékobjektum*, amely egy adott tranzakció csalásfelderítési állapotát zárja egységbe. Strukturált módszert biztosít a tranzakcióhoz kapcsolódó csalás kockázatának elemzésére és értékelésére különböző kockázati tényezők alapján.

```ruby
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
```

A `FraudDetection` osztály a következő attribútumokkal rendelkezik:

- `transaction`: A vizsgált tranzakcióra mutató referencia, amelyet csalás szempontjából elemzünk.
- `risk_factors`: Egy tömb, amely a tranzakcióhoz kapcsolódó kockázati tényezőket tárolja. Minden kockázati tényező egy hash-ként van ábrázolva, ahol a kulcs a kockázati tényező leírása, az érték pedig az adott kockázati tényezőhöz társított csalási valószínűség.

Az `add_risk_factor` metódus lehetővé teszi egy kockázati tényező hozzáadását a `risk_factors` tömbhöz. Két paramétert fogad: a `description` paramétert, ami egy string és a kockázati tényezőt írja le, valamint a `probability` paramétert, ami egy float érték és az adott kockázati tényezőhöz társított csalási valószínűséget reprezentálja. Egyszerű típusellenőrzéshez a `case...in` feltételes szerkezetet használjuk.

A lánc végén ellenőrzésre kerülő `high_risk?` predikátum metódus összehasonlítja a `fraud_probability` értéket (amit az összes kockázati tényező valószínűségének összegéből számítunk ki) a `RISK_THRESHOLD` értékkel.

A `FraudDetection` osztály tiszta és egységbe zárt módot biztosít egy tranzakció csalásfelderítésének kezelésére. Lehetővé teszi több kockázati tényező hozzáadását, mindegyiket saját leírással és valószínűséggel, valamint biztosít egy metódust annak meghatározására, hogy a tranzakció magas kockázatúnak minősül-e a számított csalási valószínűség alapján. Az osztály könnyen integrálható egy nagyobb csalásfelderítő rendszerbe, ahol különböző komponensek működhetnek együtt a csalárd tranzakciók kockázatának értékelésében és csökkentésében.

Végül, mivel ez mégiscsak egy AI-alapú programozásról szóló könyv, íme egy példa implementáció a `CheckCustomerHistory` osztályra, amely AI feldolgozást használ a Raix könyvtáram `ChatCompletion` modulját alkalmazva:

```
class CheckCustomerHistory
  include Raix::ChatCompletion

  attr_accessor :fraud_detection

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

    Consider factors such as the frequency of transactions, transaction
    amounts, geographical locations, and any deviations from the customer's
    typical behavior to generate a probability score as a float in the range
```

```
15      of 0 to 1 (with 1 being absolute certainty of fraud).
16
17      Output the results of your analysis, highlighting any red flags or areas
18      of concern in the following JSON format:
19
20      { description: <Summary of your findings>, probability: <Float> }
21    END
22
23    def self.check(fraud_detection)
24      new(fraud_detection).call
25    end
26
27    def call
28      chat_completion(json: true).tap do |result|
29        fraud_detection.add_risk_factor(**result)
30      end
31      Result.ok(fraud_detection)
32    rescue StandardError => e
33      Result.err(FraudDetectionError.new(e))
34    end
35
36    private
37
38    def initialize(fraud_detection)
39      self.fraud_detection = fraud_detection
40    end
41
42    def transcript
43      tx_history = fraud_detection.transaction.user.tx_history
44      [
45        { system: INSTRUCTION },
46        { user: "Transaction history: #{tx_history.to_json}" },
47        { assistant: "OK. Please provide the current transaction." },
48        { user: "Current transaction: #{fraud_detection.transaction.to_json}" }
49      ]
50    end
51  end
```

Ebben a példában a `CheckCustomerHistory` egy `INSTRUCTION` konstanst definiál, amely konkrét utasításokat ad az MI modellnek arra vonatkozóan, hogyan elemezze az ügyfél tranzakciós előzményeit a lehetséges csalásra utaló jelek szempontjából egy

rendszerutasításon keresztül

A `self.check` osztálymetódus létrehoz egy új `CheckCustomerHistory` példányt a `fraud_detection` objektummal, és meghívja a `call` metódust az ügyfél előzményeinek elemzéséhez.

A `call` metóduson belül az ügyfél tranzakciós előzményei lekérésre kerülnek, és egy olyan átiratba kerülnek formázásra, amely továbbításra kerül az MI modellnek. Az MI modell elemzi a tranzakciós előzményeket a megadott utasítások alapján, és visszaadja megállapításainak összefoglalását.

A megállapítások hozzáadódnak a `fraud_detection` objektumhoz, és a frissített `fraud_detection` objektum sikeres `Result`-ként kerül visszaadásra.

A `ChatCompletion` modul használatával a `CheckCustomerHistory` osztály ki tudja használni az MI erejét az ügyfél tranzakciós előzményeinek elemzéséhez és a lehetséges csalásra utaló jelek azonosításához. Ez kifinomultabb és alkalmazkodóképesebb csalásfelderítési technikákat tesz lehetővé, mivel az MI modell képes tanulni és alkalmazkodni az új mintázatokhoz és rendellenességekhez az idő során.

A frissített `FraudDetectionWorker` és a `CheckCustomerHistory` osztály szemlélteti, hogyan integrálhatók zökkenőmentesen az MI feldolgozók, intelligens elemzési és döntéshozatali képességekkel javítva a csalásfelderítési folyamatot.

Ügyfélhangulat-elemzés

Itt következik még egy hasonló példa arról, hogyan lehet implementálni egy ügyfélhangulat-elemző feldolgozót. Ezúttal sokkal kevesebb magyarázattal, mivel már valószínűleg érthető, hogyan működik ez a programozási stílus:

```
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
```

Ebben a példában a `CustomerSentimentAnalysisWorker` lépései magukban foglalják a visszajelzések előfeldolgozását (pl. zajszűrés, tokenizálás), szentimentelemzés végrehajtását a általános hangulat meghatározásához (pozitív, negatív vagy semleges), kulcskifejezések és témák kinyerését, trendek és minták azonosítását, valamint az elemzésen alapuló gyakorlati betekintések generálását.

Egészségügyi Alkalmazások

Az egészségügyi területen az AI-munkatársak segíthetik az egészségügyi szakembereket és kutatókat különböző feladatokban, ami jobb betegellátási eredményekhez és gyorsabb orvosi felfedezésekhez vezet. Néhány példa:

Betegfelvétel

Az AI-munkatársak egyszerűsíthetik a betegfelvételi folyamatot különböző feladatok automatizálásával és intelligens segítségnyújtással.

Időpontfoglalás: Az AI-munkatársak kezelhetik az időpontfoglalást a betegek preferenciáinak, rendelkezésre állásának és orvosi szükségleteik sürgősségének megértésével. Beszélgetési felületeken keresztül kommunikálhatnak a betegekkel, végigvezetve őket az időpontfoglalási folyamaton, és megtalálva a legmegfelelőbb időpontokat a beteg igényei és az egészségügyi szolgáltató rendelkezésre állása alapján.

Kórtörténet Felvétele: A betegfelvétel során az AI-munkatársak segíthetnek a beteg kórtörténetének összegyűjtésében és dokumentálásában. Interaktív párbeszédet folytathatnak a betegekkel, releváns kérdéseket feltéve korábbi betegségeikről, gyógyszereikről, allergiáikról és családi kórtörténetükről. Az AI-munkatársak természetes nyelvfeldolgozási technikákat alkalmazhatnak az összegyűjtött információk értelmezésére és strukturálására, biztosítva azok pontos rögzítését a beteg elektronikus egészségügyi dokumentációjában.

Tünetek Értékelése és Osztályozása: Az AI-munkatársak elvégezhetik a kezdeti tünetértékelést azzal, hogy kérdéseket tesznek fel a betegeknek jelenlegi tüneteikről, azok időtartamáról, súlyosságáról és az esetleges kapcsolódó tényezőkről. Orvosi tudásbázisok és gépi tanulási modellek felhasználásával ezek a munkatársak elemezhetik a megadott információkat, és előzetes differenciáldiagnózisokat generálhatnak, vagy megfelelő következő lépéseket javasolhatnak, például orvosi konzultáció ütemezését vagy önellátási intézkedések ajánlását.

Biztosítási Jogosultság Ellenőrzése: Az AI-munkatársak segíthetnek a biztosítási jogosultság ellenőrzésében a betegfelvétel során. Összegyűjthetik a beteg biztosítási adatait, API-kon vagy webszolgáltatásokon keresztül kommunikálhatnak a biztosítókkal, és ellenőrizhetik a fedezeti jogosultságot és juttatásokat. Ez az automatizálás segít egyszerűsíteni a biztosítás-ellenőrzési folyamatot, csökkentve az

adminisztratív terheket és biztosítva a pontos információrögzítést.

Betegtájékoztatás és Útmutatás: Az AI-munkatársak releváns oktatási anyagokat és útmutatásokat nyújthatnak a betegeknek specifikus egészségügyi állapotuk vagy közelgő beavatkozásaik alapján. Személyre szabott tartalmat szolgáltathatnak, válaszolhatnak gyakori kérdésekre, és útmutatást adhatnak a vizsgálat előtti felkészüléshez, gyógyszerezési utasításokhoz vagy a kezelés utáni gondozáshoz. Ez segít a betegek tájékozottságának és elkötelezettségének fenntartásában az egészségügyi útjuk során.

Az AI-munkatársak betegfelvételi folyamatban való alkalmazásával az egészségügyi szervezetek növelhetik a hatékonyságot, csökkenthetik a várakozási időt és javíthatják a teljes betegélményt. Ezek a munkatársak kezelhetik a rutinfeladatokat, pontos információkat gyűjthetnek és személyre szabott segítséget nyújthatnak, lehetővé téve az egészségügyi szakemberek számára, hogy a magas színvonalú betegellátásra összpontosíthassanak.

Betegkockázat Értékelése

Az AI-munkatársak kulcsfontosságú szerepet játszhatnak a betegkockázat értékelésében különböző adatforrások elemzésével és fejlett analitikai technikák alkalmazásával.

Adatintegráció: Az AI-munkatársak összegyűjthetik és értelmezhetik a betegadatokat több forrásból, például elektronikus egészségügyi dokumentációból (EHR), orvosi képalkotásból, laboreredményekből, viselhető eszközökből és az egészség szociális meghatározóiból. Ezen információk átfogó betegprofillá való egyesítésével az AI-munkatársak holisztikus képet nyújthatnak a beteg egészségi állapotáról és kockázati tényezőiről.

Kockázati Rétegzés: Az AI-munkatársak prediktív modelleket használhatnak a betegek különböző kockázati kategóriákba sorolásához egyéni jellemzőik és egészségügyi adataik alapján. Ez a kockázati rétegzés lehetővé teszi az egészségügyi szolgáltatók

számára azon betegek priorizálását, akik azonnali figyelmet vagy beavatkozást igényelnek. Például a magas kockázatúnak azonosított betegeket meg lehet jelölni szorosabb megfigyelésre, megelőző intézkedésekre vagy korai beavatkozásra.

Személyre Szabott Kockázati Profilok: Az AI-munkatársak személyre szabott kockázati profilokat generálhatnak minden beteg számára, kiemelve a kockázati pontszámaikhoz hozzájáruló specifikus tényezőket. Ezek a profilok tartalmazhatnak betekintéseket a beteg életmódjába, genetikai hajlamaiba, környezeti tényezőibe és az egészség szociális meghatározóiba. A kockázati tényezők részletes lebontásával az AI-munkatársak segíthetnek az egészségügyi szolgáltatóknak a megelőzési stratégiák és kezelési tervek egyéni betegigényekhez való igazításában.

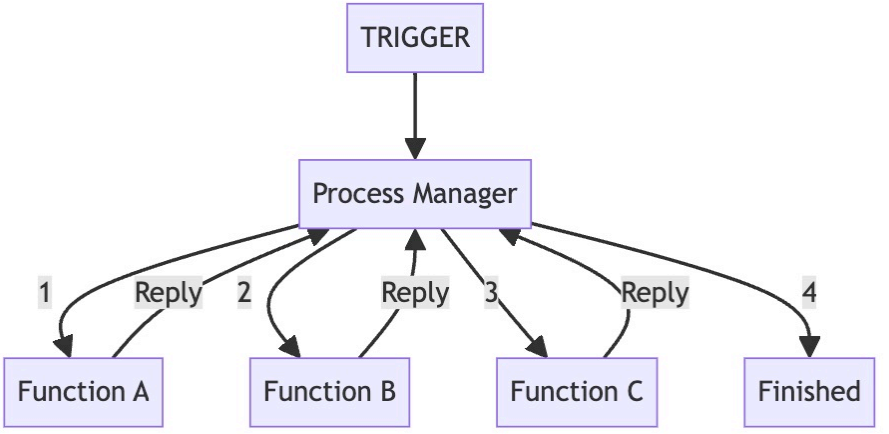
Folyamatos Kockázatfigyelés: Az AI-munkatársak folyamatosan figyelhetik a betegadatokat és valós időben frissíthetik a kockázatértékeléseket. Ahogy új információk válnak elérhetővé, például változások a vitális paraméterekben, laboreredményekben vagy a gyógyszerszedési adherenciában, az AI-munkatársak újraszámíthatják a kockázati pontszámokat és figyelmeztethetik az egészségügyi szolgáltatókat minden jelentős változásra. Ez a proaktív megfigyelés lehetővé teszi az időben történő beavatkozásokat és a betegellátási tervek módosítását.

Klinikai Döntéstámogatás: Az AI-munkatársak integrálhatják a kockázatértékelési eredményeket a klinikai döntéstámogató rendszerekbe, bizonyítékokon alapuló ajánlásokat és figyelmeztetéseket nyújtva az egészségügyi szolgáltatóknak. Például ha egy beteg kockázati pontszáma egy adott állapotra vonatkozóan meghalad egy bizonyos küszöbértéket, az AI-munkatárs javasolhatja az egészségügyi szolgáltatónak specifikus diagnosztikai tesztek, megelőző intézkedések vagy kezelési lehetőségek megfontolását a klinikai irányelvek és bevált gyakorlatok alapján.

Ezek a feldolgozók képesek nagy mennyiségű betegadatot feldolgozni, kifinomult elemzéseket végezni, és gyakorlatba átültethető betekintéseket nyújtani a klinikai döntéshozatal támogatására. Mindez végső soron jobb betegeredményekhez, csökkentett egészségügyi költségekhez és fejlettebb népegészségügyi menedzsmenthez

vezet.

AI Feldolgozó mint Folyamatkezelő



Az AI-vezérelt alkalmazások kontextusában egy feldolgozó úgy tervezhető, hogy Folyamatkezelőként működjön, ahogy azt Gregor Hohpe "Enterprise Integration Patterns" című könyve leírja. A Folyamatkezelő egy központi komponens, amely fenntartja a folyamat állapotát és meghatározza a következő feldolgozási lépéseket a közbenső eredmények alapján.

Amikor egy AI feldolgozó Folyamatkezelőként működik, egy bejövő üzenetet kap, amely elindítja a folyamatot, ezt nevezzük *indító üzenetnek*. Az AI feldolgozó ezután fenntartja a folyamat végrehajtásának állapotát (beszélgetési jegyzőkönyv formájában), és az üzenetet eszközfüggvényekként implementált feldolgozási lépések sorozatán keresztül kezeli, amelyek lehetnek szekvenciálisak vagy párhuzamosak, és saját belátása szerint hívhatók meg.

Ha olyan AI modellosztályt használsz, mint a GPT-4, amely képes párhuzamosan végrehajtani függvényeket, akkor a feldolgozód egyszerre több lépést is végrehajthat. Bevallom, én magam még nem próbáltam ezt, és az ösztönöm azt súgja, hogy az eredmények változóak lehetnek.

Minden egyes feldolgozási lépés után a vezérlés visszakerül az AI feldolgozóhoz, lehetővé téve számára, hogy meghatározza a következő feldolgozási lépés(eke)t az aktuális állapot és a kapott eredmények alapján.

Tárold az Indító Üzeneteidet

Tapasztalatom szerint okos döntés az indító üzenetet adatbázis-alapú objektumként implementálni. Így minden folyamatpéldányt egy egyedi elsődleges kulcs azonosít, és van egy hely, ahol tárolhatod a végrehajtáshoz kapcsolódó állapotot, beleértve az AI beszélgetési jegyzőkönyvét is.

Például itt van az Olympia `AccountChange` modellosztályának egyszerűsített verziója, amely egy felhasználói fiók módosítására vonatkozó kérést reprezentál.

```
# == 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

  after_commit -> {
    broadcast(:account_change_requested, self)
  }, on: :create

  state_machine initial: :requested do
    event :completed do
      transition all => :complete
    end
    event :failed do
      transition all => :requires_human_review
    end
  end
end
```

Az AccountChange osztály triggermessage-ként szolgál, amely elindítja a fiókváltoztatási kérelem kezelésének folyamatát. Érdemes megfigyelni, hogyan kerül közvetítésre az Olympia Wisper-alapú publikálás-feliratkozás alrendszerében azt követően, hogy a létrehozási tranzakció befejeződik.

A triggermessage adatbázisban történő tárolása biztosítja minden fiókváltoztatási kérelem tartós nyilvántartását. Az AccountChange osztály minden példánya egyedi elsődleges kulcsot kap, ami lehetővé teszi az egyes kérelmek egyszerű azonosítását és nyomon követését. Ez különösen hasznos az auditálási naplózás szempontjából, mivel lehetővé teszi a rendszer számára az összes fiókváltoztatás történeti nyilvántartását, beleértve a kérelmek időpontját, a kért változtatásokat és az egyes kérelmek aktuális állapotát.

A bemutatott példában az AccountChange osztály olyan mezőket tartalmaz, mint a description a kért változtatás részleteinek rögzítésére, a state a kérelem aktuális

állapotának jelzésére (például: requested, complete, requires_human_review), és a `transcript` a kérelemhez kapcsolódó AI beszélgetés átiratának tárolására. A `description` mező tartalmazza azt a tényleges promptot, amely az AI-val történő első chat completion elindításához használatos. Ezen adatok tárolása értékes kontextust biztosít, és lehetővé teszi a fiókváltoztatási folyamat jobb nyomon követését és elemzését.

A triggermessage-ek adatbázisban történő tárolása robusztus hibakezelést és helyreállítást tesz lehetővé. Ha hiba lép fel egy fiókváltoztatási kérelem feldolgozása során, a rendszer hibásként jelöli meg a kérelmet, és olyan állapotba helyezi át, amely emberi beavatkozást igényel. Ez biztosítja, hogy egyetlen kérelem se vesszen el vagy merüljön feledésbe, és minden probléma megfelelően kezelhető és megoldható legyen.

Az AI worker mint Process Manager központi irányítási pontot biztosít, és hatékony folyamatjelentési és hibakeresési képességeket tesz lehetővé. Fontos azonban megjegyezni, hogy az AI worker Process Managerként történő használata nem feltétlenül indokolt az alkalmazás minden munkafolyamatához.

AI Workerek Integrálása az Alkalmazás Architektúrájába

Az AI workerek alkalmazásarchitektúrába történő integrálásakor számos technikai szempontot kell figyelembe venni a zökkenőmentes integráció és az AI workerek, valamint egyéb alkalmazáskomponensek közötti hatékony kommunikáció biztosítása érdekében. Ez a szakasz a interfészek tervezésének, az adatáramlás kezelésének és az AI workerek életciklusának kezelésének kulcsfontosságú szempontjait tárgyalja.

Egyértelmű Interfészek és Kommunikációs Protokollok Tervezése

Az AI workerek és egyéb alkalmazáskomponensek közötti zökkenőmentes integráció érdekében elengedhetetlen az egyértelmű interfészek és kommunikációs protokollok meghatározása. Érdemes megfontolni a következő megközelítéseket:

API-alapú Integráció: Az AI workerek funkcionalitásának elérhetővé tétele jól definiált API-kon keresztül, például RESTful végpontokon vagy GraphQL sémákon keresztül. Ez lehetővé teszi más komponensek számára az AI workerekkel való interakciót standard HTTP kérések és válaszok használatával. Az API-alapú integráció egyértelmű szerződést biztosít az AI workerek és a felhasználó komponensek között, megkönnyítve az integrációs pontok fejlesztését, tesztelését és karbantartását.

Üzenetalapú Kommunikáció: Üzenetalapú kommunikációs minták implementálása, például üzenetsorok vagy publikálás-feliratkozás rendszerek használatával, az AI workerek és más komponensek közötti aszinkron interakció lehetővé tételére. Ez a megközelítés szétválasztja az AI workereket az alkalmazás többi részétől, jobb skálázhatóságot, hibatűrést és laza csatolást biztosítva. Az üzenetalapú kommunikáció különösen hasznos, amikor az AI workerek által végzett feldolgozás időigényes vagy erőforrás-intenzív, mivel lehetővé teszi az alkalmazás többi részének folyamatos működését anélkül, hogy meg kellene várni az AI workerek feladatainak befejezését.

Eseményvezérelt Architektúra: A rendszer eseményekre és triggerekre épülő tervezése, amelyek akkor aktiválják az AI workereket, amikor specifikus feltételek teljesülnek. Az AI workerek feliratkozhatnak a releváns eseményekre és reagálhatnak azokra, végrehajtva kijelölt feladataikat az események bekövetkezésekor. Az eseményvezérelt architektúra lehetővé teszi a valós idejű feldolgozást és az AI workerek igény szerinti meghívását, csökkentve a szükségtelen erőforrás-felhasználást. Ez a megközelítés különösen alkalmas olyan helyzetekben, ahol az AI workereknek specifikus műveletekre vagy az alkalmazás állapotának változásaira kell reagálniuk.

Adatáramlás és Szinkronizáció Kezelése

Az AI workerek alkalmazásba történő integrálásakor kulcsfontosságú a zökkenőmentes adatáramlás biztosítása és az adatkonzisztencia fenntartása az AI workerek és más komponensek között. Vegye figyelembe a következő szempontokat:

Adatelőkészítés: Mielőtt az adatokat az AI workerekbe táplálnánk, különböző adatelőkészítési feladatokat kell végrehajtani, például az adatok tisztítását, formázását és/vagy átalakítását. Nemcsak azt kell biztosítani, hogy az AI workerek hatékonyan feldolgozhassák az adatokat, hanem azt is, hogy ne pazaroljunk tokeneket olyan információkra való figyelemmel, amelyeket a worker legjobb esetben haszontalannak, legrosszabb esetben zavarónak találhat. Az adatelőkészítés magában foglalhatja a zaj eltávolítását, a hiányzó értékek kezelését vagy az adattípusok konvertálását.

Adatperzisztencia: Hogyan tároljuk és őrizzük meg az AI workerekbe be- és kiáramló adatokat? Vegye figyelembe olyan tényezőket, mint az adatmennyiség, a lekérdezési minták és a skálázhatóság. Szükséges-e az AI átiratának megőrzése a "gondolkodási folyamat" auditálási vagy hibakeresési célú dokumentálásához, vagy elegendő csak az eredmények nyilvántartása?

Adatlekérés: A munkavégzők által igényelt adatok beszerzése magában foglalhatja az adatbázisok lekérdezését, fájlok olvasását vagy külső API-k elérését. Fontos figyelembe venni a késleltetést és azt, hogy az MI munkavégzők hogyan férhetnek hozzá a legfrissebb adatokhoz. Szükségük van-e teljes adatbázis-hozzáférésre, vagy szűken kellene meghatározni a hozzáférésük körét a feladatuk alapján? Mi a helyzet a skálázhatósággal? Érdemes megfontolni a gyorsítótárazási mechanizmusokat a teljesítmény javítása és az alapvető adatforrások terhelésének csökkentése érdekében.

Adatszinkronizáció: Amikor több komponens, beleértve az MI munkavégzőket is, közös adatokhoz fér hozzá és módosítja azokat, fontos megfelelő szinkronizációs mechanizmusokat implementálni az adatkonzisztencia fenntartása érdekében. Az adatbázis zárolási stratégiák, mint például az optimista vagy pesszimista zárolás,

segíthetnek megelőzni a konfliktusokat és biztosítani az adatok integritását. Implementáljon tranzakciókezelési technikákat a kapcsolódó adatműveletek csoportosításához és az atomicitás, konzisztencia, izoláció és tartósság (ACID) tulajdonságok fenntartásához.

Hibakezelés és helyreállítás: Implementáljon robusztus hibakezelési és helyreállítási mechanizmusokat az adatáramlási folyamat során felmerülő adatproblémák kezelésére. Kezelje elegánsan a kivételeket és biztosítson értelmes hibaüzeneteket a hibakeresés megkönnyítéséhez. Implementáljon újrapróbálkozási mechanizmusokat és tartalék stratégiákat az átmeneti hibák vagy hálózati megszakítások kezelésére. Határozzon meg egyértelmű eljárásokat az adatok helyreállítására adatsérülés vagy -vesztés esetén.

Az adatáramlási és szinkronizációs mechanizmusok gondos tervezésével és implementálásával biztosíthatja, hogy MI munkavégzői pontos, konzisztens és naprakész adatokhoz férjenek hozzá. Ez lehetővé teszi számukra, hogy hatékonyan végezzék feladataikat és megbízható eredményeket produkáljanak.

MI munkavégzők életciklusának kezelése

Fejlesszen ki egy szabványosított folyamatot az MI munkavégzők inicializálásához és konfigurálásához. Előnyben részesítem azokat a keretrendszereket, amelyek szabványosítják a beállítások, például a modellnevek, rendszerutasítások és függvénydefiníciók meghatározását. Biztosítsa, hogy az inicializálási folyamat automatizált és reprodukálható legyen a telepítés és skálázás megkönnyítése érdekében.

Implementáljon átfogó felügyeleti és naplózási mechanizmusokat az MI munkavégzők állapotának és teljesítményének nyomon követésére. Gyűjtsön olyan metrikákat, mint az erőforrás-kihasználtság, feldolgozási idő, hibaarányok és áteresztőképesség. Használjon központosított naplózó rendszereket, mint az ELK stack (Elasticsearch, Logstash, Kibana) a több MI munkavégzőtől származó naplók összegyűjtésére és elemzésére.

Építsen hibatűrést és rugalmasságot az MI munkavégző architektúrába. Implementáljon hibakezelési és helyreállítási mechanizmusokat a hibák vagy kivételek elegáns kezelésére. A Nagy Nyelvi Modellek még mindig élvonalbeli technológiának számítanak; a szolgáltatók gyakran váratlan időpontokban leállnak. Használjon újrapróbálkozási mechanizmusokat és megszakítókat a kaszkádszerű hibák megelőzésére.

MI munkavégzők komponálhatósága és vezérlése

Az MI munkavégző architektúra egyik fő előnye a komponálhatóság, amely lehetővé teszi több MI munkavégző kombinálását és vezérlését komplex problémák megoldására. Egy nagyobb feladat kisebb, kezelhetőbb részfeladatokra bontásával, ahol minden részfeladatot egy specializált MI munkavégző kezel, hatékony és rugalmas rendszereket hozhat létre. Ebben a szakaszban különböző megközelítéseket vizsgálunk az MI munkavégzők "sokaságának" komponálására és vezérlésére.

MI munkavégzők láncolása többlépcsős munkafolyamatokhoz

Sok esetben egy komplex feladat felbontható egymást követő lépések sorozatára, ahol az egyik MI munkavégző kimenete a következő bemenete lesz. Az MI munkavégzők ilyen láncolása többlépcsős munkafolyamatot vagy adatcsatornát hoz létre. A láncban minden MI munkavégző egy specifikus részfeladatra összpontosít, és a végső kimenet az összes munkavégző együttes erőfeszítésének eredménye.

Nézzünk egy példát egy Ruby on Rails alkalmazás kontextusában felhasználók által generált tartalom feldolgozására. A munkafolyamat a következő lépéseket tartalmazza, amelyek elismerhetően valószínűleg túl egyszerűek ahhoz, hogy érdemes legyen őket

ilyen módon szétbontani a valós használati esetekben, de így a példa könnyebben érthető:

1. **Szövegtisztítás**: Egy MI munkavégző, amely felelős a HTML címkék eltávolításáért, a szöveg kisbetűssé alakításáért és az Unicode normalizálás kezeléséért.

2. **Nyelvfelismerés**: Egy MI munkavégző, amely azonosítja a megtisztított szöveg nyelvét.

3. **Érzelemelemzés**: Egy MI munkavégző, amely meghatározza a szöveg érzelmi töltetét (pozitív, negatív vagy semleges) a felismert nyelv alapján.

4. **Tartalmi kategorizálás**: Egy MI munkavégző, amely természetes nyelvfeldolgozási technikákat használva előre meghatározott kategóriákba sorolja a szöveget.

Íme egy nagyon egyszerűsített példa arra, hogyan lehet ezeket az MI munkavégzőket Ruby-ban láncolni:

```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
```

Ebben a példában a `ContentProcessor` osztály a nyers szöveggel inicializálódik, és a `process` metódusban láncba fűzi az AI feldolgozókat. Minden AI feldolgozó végrehajtja a saját specifikus feladatát, és továbbadja az eredményt a láncban következő feldolgozónak. A végső kimenet egy hash, amely tartalmazza a megtisztított szöveget, a felismert nyelvet, az érzelmi töltetet és a tartalmi kategóriát.

Párhuzamos Feldolgozás Független AI Feldolgozókkal

Az előző példában az AI feldolgozók szekvenciálisan voltak láncba fűzve, ahol minden feldolgozó feldolgozza a szöveget, és továbbadja az eredményt a következő feldolgozónak. Azonban ha több olyan AI feldolgozónk van, amelyek képesek ugyanazon a bemeneten egymástól függetlenül működni, optimalizálhatjuk a munkafolyamatot azáltal, hogy párhuzamosan dolgozzuk fel őket.

Az adott forgatókönyvben, miután a `TextCleanupWorker` elvégezte a szöveg tisztítását, a `LanguageDetectionWorker`, `SentimentAnalysisWorker` és a `CategorizationWorker` mind képesek függetlenül feldolgozni a megtisztított szöveget. Ezeknek a feldolgozóknak a párhuzamos futtatásával potenciálisan csökkenthető a teljes feldolgozási idő és javítható a munkafolyamat hatékonysága.

A párhuzamos feldolgozás megvalósításához Ruby-ban olyan párhuzamossági technikákat használhat, mint a szálak vagy az aszinkron programozás. Íme egy példa arra, hogyan módosíthatja a `ContentProcessor` osztályt úgy, hogy az utolsó három feldolgozót párhuzamosan dolgozza fel szálak használatával:

```
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
```

Ebben az optimalizált verzióban a `concurrent-ruby` könyvtárat használjuk `Concurrent::Future` objektumok létrehozására minden független AI feldolgozóhoz. A `Future` egy olyan számítást reprezentál, amely aszinkron módon, külön szálban kerül végrehajtásra.

A szöveg tisztítási lépés után három `Future` objektumot hozunk létre: `language_future`, `sentiment_future` és `category_future`. Minden `Future` a megfelelő AI feldolgozót (`LanguageDetectionWorker`, `SentimentAnalysisWorker` és `CategorizationWorker`) külön szálban futtatja, a `cleaned_text`-et adva bemenetként.

A `value` metódus meghívásával minden `Future`-nél megvárjuk a számítás befejezését és lekérjük az eredményt. A `value` metódus blokkolja a végrehajtást, amíg az eredmény nem áll rendelkezésre, így biztosítva, hogy minden párhuzamos feldolgozó befejezze a munkáját, mielőtt továbblépnénk.

Végül létrehozzuk a kimeneti hash-t a tisztított szöveggel és a párhuzamos feldolgozók eredményeivel, ugyanúgy, mint az eredeti példában.

A független AI feldolgozók párhuzamos futtatásával potenciálisan csökkenthető a teljes feldolgozási idő a szekvenciális futtatáshoz képest. Ez az optimalizálás különösen előnyös időigényes feladatok esetén vagy nagy mennyiségű adat feldolgozásakor.

Fontos azonban megjegyezni, hogy a tényleges teljesítménynyereség különböző

tényezőktől függ, mint például az egyes feldolgozók komplexitása, a rendelkezésre álló rendszererőforrások és a szálkezelés többletterhelése. Mindig jó gyakorlat a kód teljesítményének mérése és profilozása, hogy meghatározzuk az optimális párhuzamosítási szintet az adott használati esethez.

Továbbá, párhuzamos feldolgozás implementálásakor figyelembe kell venni a feldolgozók közötti megosztott erőforrásokat vagy függőségeket. Biztosítani kell, hogy a feldolgozók egymástól függetlenül, konfliktusok vagy versenyhelyzetek nélkül működhessenek. Ha vannak függőségek vagy megosztott erőforrások, megfelelő szinkronizációs mechanizmusokat kell implementálni az adatok integritásának megőrzése és a holtpontok vagy inkonzisztens eredmények elkerülése érdekében.

A Ruby Globális Értelmező Zár és az Aszinkron Feldolgozás

Fontos megérteni a Ruby Globális Értelmező Zár (GIL) következményeit, amikor Ruby-ban aszinkron szálalapú feldolgozást tervezünk.

A GIL egy olyan mechanizmus a Ruby értelmezőben, amely biztosítja, hogy egyszerre csak egy szál futtathasson Ruby kódot, még többmagos processzorokon is. Ez azt jelenti, hogy bár több szál is létrehozható és kezelhető egy Ruby folyamaton belül, egyszerre csak egy szál futtathat aktívan Ruby kódot.

A GIL-t úgy tervezték, hogy egyszerűsítse a Ruby értelmező implementációját és szálbiztonságot biztosítson a Ruby belső adatstruktúráihoz. Azonban ez korlátozza a Ruby kód valódi párhuzamos végrehajtásának lehetőségét.

Amikor szálakat használunk Ruby-ban, például a `concurrent-ruby` könyvtárral vagy a beépített `Thread` osztállyal, a szálak a GIL korlátozásainak vannak alávetve. A GIL minden szálnak rövid időszeleteket engedélyez a Ruby kód végrehajtására, mielőtt egy másik szálra váltana, így keltve a párhuzamos végrehajtás illúzióját.

Azonban a GIL miatt a Ruby kód tényleges végrehajtása szekvenciális marad. Amíg egy szál Ruby kódot futtat, a többi szál gyakorlatilag szünetel, várva a sorára, hogy megszerezhesse a GIL-t és végrehajthasson.

Ez azt jelenti, hogy a szálalapú aszinkron feldolgozás Ruby-ban leginkább I/O-kötött feladatoknál hatékony, például külső API válaszok várakozásánál (mint például külső szolgáltatású nagy nyelvi modellek) vagy fájl I/O műveleteknél. Amikor egy szál I/O műveletet végez, felszabadíthatja a GIL-t, lehetővé téve más szálak futását, amíg az I/O művelet befejeződik.

Másrészt, CPU-kötött feladatoknál, mint például intenzív számítások vagy hosszan futó AI feldolgozó műveletek, a GIL korlátozhatja a szálalapú párhuzamosítás potenciális teljesítménynyereségét. Mivel egyszerre csak egy szál futtathat Ruby kódot, a teljes végrehajtási idő nem feltétlenül csökken jelentősen a szekvenciális feldolgozáshoz képest.

A valódi párhuzamos végrehajtás eléréséhez CPU-kötött feladatoknál Ruby-ban érdemes lehet alternatív megközelítéseket megfontolni, például:

- Folyamatalapú párhuzamosítás használata több Ruby folyamattal, amelyek mindegyike külön CPU magon fut.
- Külső könyvtárak vagy keretrendszerek használata, amelyek natív kiterjesztéseket vagy interfészeket biztosítanak GIL nélküli nyelvekhez, mint a C vagy a Rust.,
- Elosztott számítási keretrendszerek vagy üzenetsorok használata a feladatok több gép vagy folyamat közötti elosztásához.

Kulcsfontosságú a feladatok természetének és a GIL által okozott korlátozások figyelembevétele az aszinkron feldolgozás tervezésekor és implementálásakor Ruby-ban. Míg a szálalapú aszinkron feldolgozás előnyös lehet I/O-kötött feladatoknál, a GIL korlátozásai miatt nem feltétlenül nyújt jelentős teljesítményjavulást CPU-kötött feladatoknál.

Együttes Technikák a Pontosság Javítására

Az együttes technikák több AI feldolgozó kimenetének kombinálását jelentik a rendszer általános pontosságának vagy robusztusságának javítása érdekében. Ahelyett, hogy egyetlen AI feldolgozóra támaszkodnánk, az együttes technikák több feldolgozó kollektív intelligenciáját használják fel a megalapozottabb döntéshozatalhoz.

Az együttes módszerek különösen fontosak, ha a munkafolyamatod különböző részei más-más MI modellekkel működnek a legjobban, ami gyakoribb, mint gondolnád. Az olyan nagy teljesítményű modellek, mint a GPT-4, rendkívül drágák a kevésbé képes nyílt forráskódú lehetőségekhez képest, és valószínűleg nincs is szükség rájuk az alkalmazásod minden egyes munkafolyamat-lépéséhez.

Egy gyakori együttes technika a többségi szavazás, ahol több MI munkavégző egymástól függetlenül dolgozza fel ugyanazt a bemenetet, és a végső kimenet a többségi konszenzus alapján kerül meghatározásra. Ez a megközelítés segíthet csökkenteni az egyedi munkavégzők hibáinak hatását és javíthatja a rendszer általános megbízhatóságát.

Nézzünk egy példát, ahol három MI munkavégzőnk van érzelemelemzésre, mindegyik különböző modellt használ vagy különböző kontextussal van ellátva. Többségi szavazással kombinálhatjuk a kimeneteiket, hogy meghatározzuk a végső érzelmi előrejelzést.

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

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

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

  end
end
```

Ebben a példában a `SentimentAnalysisEnsemble` osztály inicializálódik a szöveggel, és három különböző szentimentelemző MI munkást hív meg. Az `analyze` metódus összegyűjti az előrejelzéseket minden munkástól, és meghatározza a többségi szentimentet a `group_by` és `max_by` metódusok használatával. A végső kimenet az a szentiment, amely a legtöbb szavazatot kapja a munkások együttesétől.

Az együttesek egyértelműen olyan esetek, ahol érdemes lehet időt fektetni a párhuzamosítással való kísérletezésbe.

MI Munkások Dinamikus Kiválasztása és Meghívása

Számos, ha nem a legtöbb esetben, a konkrét MI munkás meghívása függhet futásidejű feltételektől vagy felhasználói bemenetektől. Az MI munkások dinamikus kiválasztása és meghívása rugalmasságot és alkalmazkodóképességet biztosít a rendszerben.

Könnyen kísértésbe eshetünk, hogy sok funkcionalitást próbáljunk egyetlen MI munkásba zsúfolni, sok funkcióval és egy bonyolult prompttal, ami elmagyarázza, hogyan kell ezeket meghívni. Állj ellen a kísértésnek, hidd el nekem. Az egyik oka annak, hogy az ebben a fejezetben tárgyalt megközelítést "Munkások Sokaságának" nevezzük, az az, hogy emlékeztessen minket: kívánatos sok specializált munkással rendelkezni, amelyek mind a saját kis feladatukat végzik a nagyobb cél érdekében.

Például vegyünk egy chatbot alkalmazást, ahol különböző MI munkások felelősek a különböző típusú felhasználói kérések kezeléséért. A felhasználó bemenetétől függően az alkalmazás dinamikusan választja ki a megfelelő MI munkást a kérés feldolgozásához.

```
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
```

Ebben a példában a `ChatbotController` egy felhasználói lekérdezést kap a `process_query` művelet során. Először egy `QueryClassifierWorker`-t használ a lekérdezés típusának meghatározására. A besorolt lekérdezéstípus alapján a vezérlő dinamikusan kiválasztja a megfelelő AI workert a válasz generálásához. Ez a dinamikus

kiválasztás lehetővé teszi, hogy a chatbot különböző típusú lekérdezéseket kezeljen, és a megfelelő AI workerekhez irányítsa őket.

Mivel a `QueryClassifierWorker` munkája viszonylag egyszerű, és nem igényel sok kontextust vagy függvénydefiníciót, valószínűleg implementálható egy ultra-gyors kis LLM segítségével, mint például a `mistralai/mixtral-8x7b-instruct:nitro`. Képességei sok feladat esetén megközelítik a GPT-4 szintjét, és amikor ezt írom, a Groq villámgyors, 444 token/másodperces sebességgel tudja kiszolgálni.

Hagyományos NLP és LLM-ek kombinálása

Bár a Nagy Nyelvi Modellek (LLM) forradalmasították a természetes nyelvfeldolgozás (NLP) területét, páratlan sokoldalúságot és teljesítményt nyújtva számos feladatban, nem mindig jelentik a leghatékonyabb vagy költséghatékonyabb megoldást minden problémára. Sok esetben a hagyományos NLP technikák és az LLM-ek kombinálása optimalizáltabb, célzottabb és gazdaságosabb megközelítést eredményezhet specifikus NLP kihívások megoldására.

Gondoljunk az LLM-ekre úgy, mint a természetes nyelvfeldolgozás svájci bicskájára – hihetetlenül sokoldalúak és erőteljesek, de nem feltétlenül a legjobb eszközök minden feladathoz. Néha egy dedikált eszköz, mint egy dugóhúzó vagy konzervnyitó, hatékonyabb lehet egy adott feladathoz. Hasonlóképpen, a hagyományos NLP technikák, mint például a dokumentumklaszterezés, témaazonosítás és osztályozás, gyakran célzottabb és költséghatékonyabb megoldásokat kínálhatnak az NLP-folyamat bizonyos aspektusaihoz.

A hagyományos NLP technikák egyik fő előnye a számítási hatékonyságuk. Ezek a módszerek, amelyek gyakran egyszerűbb statisztikai modelleken vagy szabályalapú megközelítéseken alapulnak, sokkal gyorsabban és kisebb számítási többlettel tudnak

nagy mennyiségű szöveges adatot feldolgozni az LLM-ekhez képest. Ez különösen alkalmassá teszi őket olyan feladatokra, amelyek nagy dokumentumkorpuszok elemzését és rendszerezését igénylik, például hasonló cikkek klaszterezését vagy kulcstémák azonosítását szöveggyűjteményekben.

Ráadásul a hagyományos NLP technikák gyakran magas pontosságot érhetnek el specifikus feladatokban, különösen amikor domain-specifikus adathalmazokon tanítják őket. Például egy jól hangolt dokumentumosztályozó, amely hagyományos gépi tanulási algoritmusokat használ, mint a Tartóvektor-gépek (SVM) vagy a naiv Bayes, minimális számítási költséggel pontosan kategorizálhat dokumentumokat előre meghatározott kategóriákba.

Az LLM-ek azonban igazán akkor ragyognak, amikor a nyelv, a kontextus és a következtetés mélyebb megértésére van szükség. Képességük a koherens és kontextuálisan releváns szöveg generálására, kérdések megválaszolására és hosszú szakaszok összefoglalására felülmúlhatatlan a hagyományos NLP módszerekhez képest. Az LLM-ek hatékonyan kezelik a komplex nyelvi jelenségeket, mint például a kétértelműséget, a koreferenciát és az idiomatikus kifejezéseket, így nélkülözhetetlenek olyan feladatokhoz, amelyek természetes nyelvgenerálást vagy -megértést igényelnek.

Az igazi erő a hagyományos NLP technikák és az LLM-ek kombinálásában rejlik, olyan hibrid megközelítések létrehozásával, amelyek mindkét módszer erősségeit kihasználják. A hagyományos NLP módszerek használatával a dokumentum-előfeldolgozás, klaszterezés és témaextrakció terén hatékonyan szervezheti és strukturálhatja a szöveges adatokat. Ez a strukturált információ aztán LLM-ekbe táplálható fejlettebb feladatokhoz, mint például összefoglalók generálása, kérdések megválaszolása vagy átfogó jelentések készítése.

Vegyünk például egy olyan esetet, ahol egy specifikus terület trendjeiről szeretne jelentést készíteni egy nagy, egyedi trenddokumentumokból álló korpusz alapján. Ahelyett, hogy kizárólag LLM-ekre támaszkodna, ami számításilag költséges és időigényes lehet nagy mennyiségű szöveg feldolgozásakor, alkalmazhat egy hibrid

megközelítést:

1. Használjon hagyományos NLP technikákat, mint például témamodellezést (pl. látens Dirichlet-allokáció) vagy klaszterezési algoritmusokat (pl. K-közép), a hasonló trenddokumentumok csoportosítására és a kulcsfontosságú témák azonosítására a korpuszban.
2. Táplálja be a klaszterezett dokumentumokat és az azonosított témákat egy LLM-be, kihasználva annak kiváló nyelvértési és generálási képességeit, hogy koherens és informatív összefoglalókat készítsen minden klaszterhez vagy témához.
3. Végül használja az LLM-et egy átfogó trendjelentés generálására az egyedi összefoglalók kombinálásával, kiemelve a legjelentősebb trendeket, és betekintést valamint ajánlásokat nyújtva az összesített információk alapján.

A hagyományos NLP technikák és LLM-ek ilyen módon történő kombinálásával hatékonyan feldolgozhat nagy mennyiségű szöveges adatot, kinyerhet értelmes betekintéseket, és magas minőségű jelentéseket generálhat, miközben optimalizálja a számítási erőforrásokat és költségeket.

Amikor NLP projektekbe kezdesz, elengedhetetlen, hogy gondosan értékeld az egyes feladatok specifikus követelményeit és korlátozásait, valamint megfontold, hogyan lehet a hagyományos NLP módszereket és az LLM-eket együttesen felhasználni a legjobb eredmények elérése érdekében. A hagyományos technikák hatékonyságának és pontosságának ötvözésével az LLM-ek sokoldalúságával és erejével olyan rendkívül hatékony és gazdaságos NLP megoldásokat hozhatsz létre, amelyek értéket teremtenek felhasználóid és az érintettek számára.

Eszközhasználat

Az AI-vezérelt alkalmazásfejlesztés területén az “eszközhasználat” vagy “függvényhívás” koncepciója olyan hatékony technikává nőtte ki magát, amely lehetővé teszi, hogy az LLM külső eszközökhöz, API-khoz, függvényekhez, adatbázisokhoz és egyéb erőforrásokhoz kapcsolódjon. Ez a megközelítés a szövegkimeneten túlmutató, gazdagabb viselkedéskészletet tesz lehetővé, valamint dinamikusabb interakciókat biztosít az AI komponensek és az alkalmazás ökoszisztémájának többi része között. Ahogy ebben a fejezetben látni fogjuk, az eszközhasználat azt a lehetőséget is megadja, hogy az AI modell strukturált módon generáljon adatokat.

Mi az eszközhasználat?

Az eszközhasználat, más néven függvényhívás, olyan technika, amely lehetővé teszi a fejlesztők számára, hogy meghatározzanak egy függvénylistát, amellyel az

LLM a generálási folyamat során képes interakcióba lépni. Ezek az eszközök az egyszerű segédfüggvényektől kezdve a komplex API-kig vagy adatbázis-lekérdezésekig terjedhetnek. Azáltal, hogy az LLM számára hozzáférést biztosítanak ezekhez az eszközökhöz, a fejlesztők kibővíthetik a modell képességeit, és olyan feladatok végrehajtására tehetik alkalmassá, amelyek külső tudást vagy műveleteket igényelnek.

ábra 8. Egy függvénydefiníció példája egy dokumentumokat elemző AI munkához

```
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
```

Az eszközhasználat mögött álló kulcsfontosságú gondolat az, hogy az LLM-nek megadjuk azt a képességet, hogy dinamikusan kiválassza és végrehajtsa a megfelelő eszközöket a felhasználó bevitele vagy az adott feladat alapján. Ahelyett, hogy kizárólag

a modell előre betanított tudására támaszkodnánk, az eszközhasználat lehetővé teszi az LLM számára, hogy külső erőforrásokat használjon fel pontosabb, relevánsabb és végrehajtható válaszok generálásához. Az eszközhasználat olyan technikákat, mint a RAG (Lekérdezéssel Bővített Generálás), sokkal könnyebben megvalósíthatóvá tesz, mint egyébként lennének.

> Megjegyzendő, hogy hacsak másképp nem jelezzük, ez a könyv feltételezi, hogy az AI modellje nem fér hozzá beépített szerveroldali eszközökhöz. Minden eszközt, amit az AI számára elérhetővé szeretne tenni, Önnek kifejezetten deklarálnia kell minden API kérésben, valamint gondoskodnia kell annak végrehajtásáról, ha és amikor az AI jelzi, hogy használni szeretné azt az eszközt a válaszában.

Az Eszközhasználat Potenciálja

Az eszközhasználat széles körű lehetőségeket nyit meg az AI-vezérelt alkalmazások számára. Íme néhány példa arra, hogy mi érhető el az eszközhasználattal:

1. **Chatbotok és Virtuális Asszisztensek**: Azáltal, hogy egy LLM-et külső eszközökhöz kapcsolunk, a chatbotok és virtuális asszisztensek összetettebb feladatokat is képesek végrehajtani, például információk lekérése adatbázisokból, API hívások végrehajtása vagy más rendszerekkel való interakció. Például egy chatbot használhat CRM eszközt egy üzlet státuszának megváltoztatására a felhasználó kérése alapján.
2. **Adatelemzés és Betekintések**: Az LLM-ek összekapcsolhatók adatelemző eszközökkel vagy könyvtárakkal fejlett adatfeldolgozási feladatok végrehajtásához. Ez lehetővé teszi az alkalmazások számára, hogy betekintéseket generáljanak, összehasonlító elemzéseket végezzenek, vagy adatvezérelt ajánlásokat nyújtsanak felhasználói lekérdezések alapján.

3. **Keresés és Információlekérés**: Az eszközhasználat lehetővé teszi az LLM-ek számára a keresőmotorokkal, vektoradatbázisokkal vagy más információlekérő rendszerekkel való interakciót. A felhasználói lekérdezések keresési lekérdezésekké alakításával az LLM több forrásból is képes releváns információkat lekérni és átfogó válaszokat adni a felhasználói kérdésekre.
4. **Integráció Külső Szolgáltatásokkal**: Az eszközhasználat lehetővé teszi az AI-vezérelt alkalmazások és külső szolgáltatások vagy API-k közötti zökkenőmentes integrációt. Például egy LLM képes lehet időjárási API-val kommunikálni valós idejű időjárás-frissítések biztosításához, vagy fordítási API-val többnyelvű válaszok generálásához.

Az Eszközhasználat Munkafolyamata

Az eszközhasználat munkafolyamata jellemzően négy kulcsfontosságú lépést tartalmaz:

1. Függvénydefiníciók beillesztése a kérés kontextusába
2. Dinamikus (vagy explicit) eszközválasztás
3. Függvény(ek) végrehajtása
4. A eredeti prompt opcionális folytatása

Nézzük át részletesen mindegyik lépést.

Függvénydefiníciók beillesztése a kérés kontextusába

Az AI tudja, hogy milyen eszközök állnak rendelkezésére, mert Ön egy listát ad neki a kiegészítési kérés részeként (általában JSON séma változatként definiált függvények formájában).

A eszközdefiníció pontos szintaxisa modellfüggő.

Így definiálhatunk egy `get_weather` függvényt a Claude 3-ban:

```
{
    "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"]
    }
}
```

És így definiálnád ugyanezt a függvényt GPT-4 esetében, a `tools` paraméter értékeként átadva:

```
{
    "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"],
```

```
    },
}
```

> Majdnem ugyanaz, csak valamiért mégis más! Mennyire bosszantó.

A függvénydefiníciók meghatározzák a nevet, a leírást és a bemeneti paramétereket. A bemeneti paraméterek további meghatározására használhatók olyan attribútumok, mint a felsorolások az elfogadható értékek korlátozására, valamint annak meghatározása, hogy egy paraméter kötelező-e vagy sem.

A tényleges függvénydefiníciókon túl a rendszerutasításban útmutatásokat vagy kontextust is megadhatsz arra vonatkozóan, hogy miért és hogyan kell használni a függvényt.

Például az Olympiában található Webes keresőeszközöm tartalmazza ezt a rendszerutasítást, amely emlékezteti az MI-t arra, hogy a említett eszközök a rendelkezésére állnak:

```
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.
```

A részletes leírások biztosítása tekinthető a legfontosabb tényezőnek az eszköz teljesítményében. A leírásoknak minden részletre ki kell térniük az eszközzel kapcsolatban, beleértve:

- Mit csinál az eszköz
- Mikor kell használni (és mikor nem)
- Mit jelent minden paraméter és hogyan befolyásolja az eszköz működését

- Minden fontos megszorítást vagy korlátozást, ami az eszköz implementációjára vonatkozik

Minél több kontextust tudsz adni az AI-nak az eszközeidről, annál jobban fog teljesíteni a használatuk időzítésének és módjának meghatározásában. Például az Anthropic legalább 3-4 mondatot javasol eszközönként a Claude 3 sorozatához, még többet, ha az eszköz összetett.

Bár nem feltétlenül intuitív, de a leírásokat fontosabbnak tekintik, mint a példákat. Míg beilleszthetsz példákat az eszköz használatára vonatkozóan annak leírásába vagy a kísérő promptba, ez kevésbé fontos, mint az eszköz céljának és paramétereinek világos és átfogó magyarázata. Csak akkor adj hozzá példákat, miután teljes mértékben kidolgoztad a leírást.

Íme egy Stripe-szerű API függvény specifikációjának példája:

```
{
  "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" ]
      }
    }
}
```

A gyakorlatban néhány modellnek gondot okoz a beágyazott függvényspecifikációk kezelése és a komplex kimeneti adattípusok, mint például a tömbök, szótárak stb. kezelése. Elméletben azonban tetszőleges mélységű JSON Schema specifikációkat kellene tudni megadni!

Dinamikus eszközválasztás

Amikor olyan csevegési kiegészítést hajt végre, amely eszközmeghatározásokat tartalmaz, az LLM dinamikusan kiválasztja a legmegfelelőbb eszköz(öke)t, és generálja a szükséges bemeneti paramétereket minden eszközhöz.

A gyakorlatban az AI képessége arra, hogy *pontosan* a megfelelő függvényt hívja meg, és *pontosan* kövesse a bemeneti paraméterek specifikációját, változó sikerességű. A hőmérséklet hiperparaméter 0.0-ra állítása sokat segít, de tapasztalatom szerint még így is előfordulnak alkalmanként hibák. Ezek a hibák között szerepelnek hallucináló függvénynevek, rosszul megnevezett vagy egyszerűen hiányzó bemeneti paraméterek. A paraméterek JSON formátumban kerülnek átadásra, ami azt jelenti, hogy néha hibákat láthat, amelyeket csonkolt, rosszul idézett vagy egyéb módon sérült JSON okoz.

Az Önjavító adatok minták segíthetnek automatikusan kijavítani a szintaktikai hibák miatt megszakadt függvényhívásokat.

Kényszerített (más néven explicit) eszközválasztás

Néhány modell lehetőséget ad arra, hogy kikényszerítse egy adott függvény hívását a kérés paramétereként. Egyébként teljesen az AI belátására van bízva, hogy meghívja-e a függvényt vagy sem.

A függvényhívás kikényszerítésének képessége kulcsfontosságú bizonyos forgatókönyvekben, ahol biztosítani szeretné, hogy egy adott eszköz vagy függvény végrehajtódjon, függetlenül az AI dinamikus kiválasztási folyamatától. Több okból is fontos ez a képesség:

1. **Explicit irányítás**: Előfordulhat, hogy az AI-t *Diszkrét komponensként* vagy előre meghatározott munkafolyamatban használja, amely megköveteli egy adott függvény végrehajtását egy adott időpontban. A hívás kikényszerítésével

garantálhatja, hogy a kívánt függvény kerül meghívásra, ahelyett, hogy szépen meg kellene kérnie az AI-t erre.

2. **Hibakeresés és tesztelés**: Az AI-vezérelt alkalmazások fejlesztése és tesztelése során a függvényhívások kikényszerítésének képessége felbecsülhetetlen értékű a hibakeresés szempontjából. A specifikus függvények explicit meghívásával elkülönítheti és tesztelheti az alkalmazás egyes komponenseit. Ez lehetővé teszi a függvényimplementációk helyességének ellenőrzését, a bemeneti paraméterek validálását és a várt eredmények visszatérésének biztosítását.
3. **Szélsőséges esetek kezelése**: Lehetnek olyan szélsőséges esetek vagy kivételes helyzetek, ahol az AI dinamikus kiválasztási folyamata esetleg nem választja ki egy olyan függvény végrehajtását, amelyet kellene, és ezt Ön külső folyamatok alapján tudja. Ilyen esetekben a függvényhívás kikényszerítésének lehetősége lehetővé teszi ezeknek a helyzeteknek a explicit kezelését. Definiáljon szabályokat vagy feltételeket az alkalmazás logikájában annak meghatározására, hogy mikor írja felül az AI döntését.
4. **Konzisztencia és reprodukálhatóság**: Ha konkrét függvények sorozatát kell végrehajtani egy meghatározott sorrendben, a hívások kikényszerítése garantálja, hogy minden alkalommal ugyanazt a sorrendet követi. Ez különösen fontos olyan alkalmazásokban, ahol a konzisztencia és a kiszámítható viselkedés kritikus, például pénzügyi rendszerekben vagy tudományos szimulációkban.
5. **Teljesítményoptimalizálás**: Bizonyos esetekben a függvényhívás kikényszerítése teljesítményoptimalizáláshoz vezethet. Ha tudja, hogy egy adott feladathoz egy konkrét függvényre van szükség, és hogy az AI dinamikus kiválasztási folyamata szükségtelen többletterhelést okozhat, megkerülheti a kiválasztási folyamatot és közvetlenül meghívhatja a szükséges függvényt. Ez segíthet csökkenteni a késleltetést és javítani az alkalmazás általános hatékonyságát.

Összefoglalva, a függvényhívások kikényszerítésének képessége az AI-vezérelt alkalmazásokban explicit irányítást biztosít, segít a hibakeresésben és tesztelésben, kezeli a szélsőséges eseteket, biztosítja a konzisztenciát és reprodukálhatóságot. Ez egy

hatékony eszköz az arzenálban, de meg kell tárgyalnunk még egy aspektusát ennek a fontos funkciónak.

Sok döntéshozatali esetben azt szeretnénk, hogy a modell mindig végrehajtson egy függvényhívást, és soha ne válaszoljon csak a belső tudása alapján. Például, ha több, különböző feladatokra specializálódott modell között irányít (többnyelvű bemenet, matematika stb.), használhatja a függvényhívó modellt a kérések delegálására valamelyik segédmodellhez, és soha nem válaszol önállóan.

Eszközválasztási paraméter

A GPT-4 és más nyelvi modellek, amelyek támogatják a függvényhívást, egy `tool_choice` paramétert biztosítanak annak szabályozására, hogy szükséges-e eszközhasználat a kiegészítés részeként. Ennek a paraméternek három lehetséges értéke van:

- az `auto` teljes döntési szabadságot ad az AI-nak egy eszköz használatára vagy egyszerű válaszadásra
- a `required` közli az AI-val, hogy *muszáj* egy eszközt hívnia válaszadás *helyett*, de az eszköz kiválasztását az AI-ra bízza
- A harmadik lehetőség a kikényszeríteni kívánt `függvény_neve` paraméter beállítása. Erről bővebben a következő részben.

Vedd figyelembe, hogy ha a tool choice értékét `required`-re állítod, a modell kénytelen lesz kiválasztani a legmegfelelőbb függvényt a rendelkezésre álló függvények közül, még akkor is, ha valójában egyik sem illik tökéletesen a prompthoz. A publikáció időpontjában nem ismerek olyan modellt, amely üres `tool_calls` választ adna vissza, vagy más módon jelezné, hogy nem talált megfelelő függvényt a híváshoz.

Függvény Kényszerítése Strukturált Kimenet Érdekében

A függvényhívás kikényszerítésének képessége lehetőséget ad arra, hogy strukturált adatokat kényszeríts ki egy chat completionből ahelyett, hogy neked kellene kinyerned azokat a szöveges válaszból.

> Miért olyan nagy szó a függvények kényszerítése strukturált kimenet érdekében? Egyszerűen azért, mert a strukturált adatok kinyerése egy NNM kimenetéből rendkívül macerás. Megkönnyítheted az életed azzal, hogy XML formátumban kéred az adatokat, de akkor meg XML-t kell elemezned. És mit teszel, amikor az XML hiányzik, mert az MI azt válaszolta: "Sajnálom, de nem tudom létrehozni a kért adatokat, mert bla, bla, bla..."

Amikor így használsz eszközöket:

- Valószínűleg érdemes egyetlen eszközt definiálnod a kérésedben
- Ne feledd, hogy a `tool_choice` paraméterrel kényszerítsd a függvény használatát
- Ne feledd, hogy a modell továbbítja a bemenetet az eszköznek, így az eszköz neve és leírása a modell szemszögéből kell, hogy értelmes legyen, nem a tiédből

Ez az utolsó pont megérdemel egy példát a tisztázás érdekében. Tegyük fel, hogy arra kéred az MI-t, hogy végezzen hangulatértékelést a felhasználói szövegen. A függvény neve nem `analyze_sentiment` lenne, hanem inkább valami olyasmi, mint `save_sentiment_analysis`. Az MI végzi a hangulatértékelést, *nem az eszköz.* Az eszköz csak menti az elemzés eredményeit (a modell szemszögéből).

Itt egy példa arra, hogyan használhatjuk a Claude 3-at egy kép összefoglalásának jól strukturált JSON formátumban való rögzítésére, ezúttal parancssorból, `curl` használatával:

```
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."
                }
            ]
        }
    ]
}'
```

A bemutatott példában az Anthropic Claude 3 modelljét használjuk egy kép strukturált JSON összefoglalójának generálására. Így működik:

1. A kérés payload-jában egyetlen eszközt definiálunk `record_summary` néven a `tools` tömbben. Ez az eszköz felelős a kép összefoglalójának jól strukturált JSON formátumba való rögzítéséért.
2. A `record_summary` eszköz rendelkezik egy `input_schema`-val, amely meghatározza a várt JSON kimenet szerkezetét. Három tulajdonságot definiál:

    - `key_colors`: Objektumok tömbje, amely a kép főbb színeit reprezentálja. Minden színobjektum tartalmazza a vörös, zöld és kék értékeket (0.0-tól 1.0-ig terjedő tartományban), valamint egy ember által olvasható színnevet snake_case formátumban.
    - `description`: Egy string tulajdonság a kép rövid leírásához, 1-2 mondatra korlátozva.
    - `estimated_year`: Egy opcionális egész szám tulajdonság a kép készítésének becsült évéhez, amennyiben nem fiktív fotóról van szó.

3. A `messages` tömbben base64 kódolású karakterláncként adjuk meg a képadatokat a média típussal együtt. Ez lehetővé teszi a modell számára a kép feldolgozását a bemenet részeként.
4. Emellett arra kérjük a Claude-ot, hogy használja a `record_summary` eszközt a kép leírásához.
5. Amikor a kérést elküldjük a Claude 3 modellnek, az elemzi a képet és az előírt `input_schema` alapján JSON összefoglalót generál. A modell kiemeli a főbb színeket, rövid leírást ad, és megbecsüli a kép készítésének évét (ha alkalmazható).
6. A generált JSON összefoglaló a `record_summary` eszköz paramétereiként kerül átadásra, strukturált reprezentációt biztosítva a kép főbb jellemzőiről.

A `record_summary` eszköz jól definiált `input_schema`-val való használata lehetővé teszi, hogy strukturált JSON összefoglalót kapjunk egy képről anélkül, hogy egyszerű szöveges kivonatolásra támaszkodnánk. Ez a megközelítés biztosítja, hogy a kimenet következetes formátumot kövessen, és könnyen feldolgozható legyen az alkalmazás későbbi komponensei által.

Az a képesség, hogy kikényszeríthetjük egy függvény hívását és meghatározhatjuk a várt kimeneti struktúrát, az eszközhasználat erőteljes funkciója az AI-vezérelt alkalmazásokban. Ez lehetővé teszi a fejlesztők számára, hogy nagyobb kontrolljuk legyen a generált kimenet felett, és egyszerűsíti az AI által generált adatok integrációját az alkalmazásuk munkafolyamatába.

Függvény(ek) Végrehajtása

Miután definiáltad a függvényeket és utasítottad az AI-t, amely úgy döntött, hogy meg kell hívnia az egyik függvényedet, itt az ideje, hogy az alkalmazáskódod vagy a könyvtárad (ha például a `raix-rails` Ruby gemet használod) továbbítsa a függvényhívást és annak paramétereit a megfelelő implementációhoz *az alkalmazáskódodban.*

Az alkalmazáskódod dönti el, hogy mit kezdjen a függvény végrehajtásának eredményeivel. Lehet, hogy ez mindössze egyetlen sor kódot jelent egy lambdában, vagy lehet, hogy egy külső API hívását jelenti. Lehet, hogy egy másik AI komponens hívását jelenti, vagy akár több száz vagy ezer sornyi kódot a rendszered többi részében. Ez teljesen rajtad múlik.

Néha a függvényhívás a művelet vége, de ha az eredmények olyan információt képviselnek egy gondolatmenetben, amelyet az AI-nak folytatnia kell, akkor az alkalmazáskódodnak be kell illesztenie a végrehajtás eredményeit a csevegés átiratába, és hagynia kell, hogy az AI folytassa a feldolgozást.

Például itt van egy Raix függvénydeklaráció, amelyet az Olympia `AccountManager`-e használ az ügyfeleinkkel való kommunikációhoz az ügyfélszolgálat Intelligens Munkafolyamat Vezérlésének részeként.

```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
```

Lehet, hogy nem azonnal világos, mi történik itt, ezért részletesen elmagyarázom.

1. Az `AccountManager` osztály számos fiókkezeléssel kapcsolatos függvényt definiál. Képes megváltoztatni a előfizetési csomagot, hozzáadni és eltávolítani csapattagokat, és egyéb műveleteket végezni.
2. A felső szintű utasítások megmondják az `AccountManager`-nek, hogy értesítenie kell a fiók tulajdonosát a fiókváltoztatási kérelem eredményeiről a `notify_account_owner` függvény használatával.
3. A függvény tömör definíciója tartalmazza:

- nevét
- leírását
- paramétereit `message: { type: "string" }`
- egy végrehajtandó blokkot, amikor a függvényt meghívják

Miután az átirat frissült a függvényblokk eredményeivel, a `chat_completion` metódus újra meghívásra kerül. Ez a metódus felelős a frissített beszélgetési átirat visszaküldéséért az AI modellnek további feldolgozásra. Ezt a folyamatot *beszélgetési ciklusnak* nevezzük.

Amikor az AI modell új csevegés befejezési kérést kap a frissített átirattal, hozzáfér az előzőleg végrehajtott függvény eredményeihez. Elemezni tudja ezeket az eredményeket, beépítheti őket a döntéshozatali folyamatába, és generálhatja a következő választ vagy műveletet a beszélgetés összesített kontextusa alapján. Választhat további függvények végrehajtása között a frissített kontextus alapján, vagy generálhat egy végső választ az eredeti kérésre, ha úgy ítéli meg, hogy nincs szükség további függvényhívásokra.

Az Eredeti Kérés Opcionális Folytatása

Amikor visszaküldi az eszköz eredményeit az LLM-nek és folytatja az eredeti kérés feldolgozását, az AI ezeket az eredményeket használja fel további függvények hívásához vagy egy végső egyszerű szöveges válasz generálásához.

Néhány modell, mint például a Cohere Command-R modellje képes hivatkozni a válaszaiban az általuk használt konkrét eszközökre, ami további átláthatóságot és nyomonkövethetőséget biztosít.

A használt modelltől függően a függvényhívás eredményei vagy saját speciális szerepkörrel rendelkező átirat üzenetekben élnek, vagy valamilyen más szintaxisban jelennek meg. De a lényeges rész az, hogy ezek az adatok az átiratban legyenek, hogy az AI figyelembe vehesse őket, amikor eldönti, mit tegyen következőként.

Egy gyakori (és potenciálisan költséges) hibafeltétel, ha elfelejtjük hozzáadni a függvény eredményeit az átirathoz, mielőtt folytatnánk a csevegést. Ennek eredményeként az AI gyakorlatilag ugyanúgy kap promptot, mint mielőtt először meghívta volna a függvényt. Más szóval, az AI szempontjából még nem hívta meg a függvényt. Így újra meghívja. És újra. És újra, végtelenségig, amíg meg nem szakítjuk. Reméljük, hogy a kontextus nem volt túl nagy, és a modell nem volt túl drága!

Bevált Gyakorlatok az Eszközhasználathoz

Az eszközhasználat maximális kihasználásához vegye figyelembe a következő bevált gyakorlatokat.

Leíró Definíciók

Adjon meg világos és leíró neveket és leírásokat minden eszközhöz és annak bemeneti paramétereihez. Ez segíti az LLM-et jobban megérteni az egyes eszközök célját és képességeit.

> Tapasztalatból mondhatom, hogy az a közmondás, miszerint "a névadás nehéz", itt is érvényes; drámaian eltérő eredményeket láttam LLM-ektől pusztán a függvények neveinek vagy a leírások megfogalmazásának megváltoztatásával. Néha a leírások eltávolítása *javítja* a teljesítményt.

Eszközeredmények Feldolgozása

Amikor az eszköz eredményeit visszaküldi az LLM-nek, győződjön meg róla, hogy jól strukturáltak és átfogóak. Használjon értelmes kulcsokat és értékeket az egyes eszközök kimenetének reprezentálásához. Kísérletezzen különböző formátumokkal, és nézze meg, melyik működik a legjobban, a JSON-tól az egyszerű szövegig.

Az Eredmény Értelmező ezt a kihívást úgy kezeli, hogy AI-t alkalmaz az eredmények elemzésére és emberbarát magyarázatok, összefoglalók vagy kulcsfontosságú tanulságok szolgáltatására.

Hibakezelés

Implementáljon robusztus hibakezelési mechanizmusokat azon esetek kezelésére, amikor az LLM érvénytelen vagy nem támogatott bemeneti paramétereket generálhat az eszközhívásokhoz. Kezelje elegánsan és álljon helyre bármilyen hibából, ami az eszköz végrehajtása során előfordulhat.

Az AI egyik rendkívül kellemes tulajdonsága, hogy megérti a hibaüzeneteket! Ez azt jelenti, hogy ha gyors és egyszerű megoldásban gondolkodunk, egyszerűen elkaphatjuk az eszköz implementációjában generált kivételeket, és visszaküldhetjük az AI-nak, hogy tudja, mi történt!

Például, itt van az Olympiában használt Google keresés implementációjának egy egyszerűsített verziója:

```
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
```

A Google keresések az Olympiában kétlépéses folyamatként működnek. Először végrehajtod a keresést, majd összefoglalod az eredményeket. Ha bármilyen hiba történik, függetlenül annak jellegétől, a hibaüzenet becsomagolásra kerül és visszajut az AI-hoz. Ez a technika gyakorlatilag az összes *Intelligens Hibakezelési* minta alapja.

Például tegyük fel, hogy a `GoogleSearch` API hívás egy 503 Szolgáltatás Nem Elérhető kivétel miatt hibásodik meg. Ez felbuborékol a legfelső szintű kivételkezelésig, és a hiba leírása visszakerül az AI-hoz mint a függvényhívás eredménye. Ahelyett, hogy

egyszerűen egy üres képernyőt vagy technikai hibát mutatnánk a felhasználónak, az AI valami olyasmit mond, hogy "Sajnálom, de jelenleg nem férek hozzá a Google Keresési képességeimhez. Ha szeretnéd, később újrapróbálhatom."

Ez talán csak egy ügyes trükknek tűnhet, de gondolj egy másféle hibára, amikor az AI egy külső API-t hív meg és közvetlen irányítása van a továbbítandó paraméterek felett. Mi van, ha hibát vétett ezeknek a paramétereknek a generálásában? Feltéve, hogy a külső API hibaüzenete elég részletes, ha a hibaüzenetet visszajuttatjuk a hívó AI-hoz, az újragondolhatja ezeket a paramétereket és újrapróbálkozhat. Automatikusan. Függetlenül attól, hogy mi volt a hiba.

Most gondolj bele, mennyi munkába kerülne ilyen robusztus hibakezelést megvalósítani *normál* kódban. Gyakorlatilag lehetetlen.

Iteratív Finomítás

Ha az LLM nem a megfelelő eszközöket ajánlja vagy szuboptimális válaszokat generál, iterálj az eszköz definíciókon, leírásokon és bemeneti paramétereken. Folyamatosan finomítsd és javítsd az eszközök beállítását a megfigyelt viselkedés és a kívánt eredmények alapján.

1. Kezdd egyszerű eszköz definíciókkal: Kezdj olyan eszközök definiálásával, amelyeknek világos és tömör nevük, leírásuk és bemeneti paramétereik vannak. Kerüld az eszközbeállítások túlbonyolítását kezdetben, és összpontosíts az alapvető funkcionalitásra. Például ha el szeretnéd menteni a hangulatelemzés eredményeit, kezdd egy olyan alapvető definícióval, mint:

```
{
  "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. Tesztelés és megfigyelés: Miután az kezdeti eszköz definíciókat létrehoztad, teszteld őket különböző promptokkal és figyeld meg, hogyan működik együtt az LLM az eszközzel. Figyelj a generált válaszok minőségére és relevanciájára. Ha az LLM szuboptimális válaszokat generál, itt az ideje finomítani az eszköz definíciókat.
3. Leírások finomítása: Ha az LLM félreérti egy eszköz célját, próbáld meg finomítani az eszköz leírását. Adj meg több kontextust, példákat vagy magyarázatokat, hogy segítsd az LLM-et az eszköz hatékony használatában. Például frissítheted a szentimentelemző eszköz leírását, hogy konkrétabban foglalkozzon a szöveg *érzelmi tónusával*:

```
{
  "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. Bemeneti paraméterek módosítása: Ha az LLM érvénytelen vagy nem megfelelő bemeneti paramétereket generál egy eszközhöz, fontolja meg a paraméter-

definíciók módosítását. Adjon hozzá konkrétabb megkötéseket, validációs szabályokat vagy példákat a várt bemeneti formátum tisztázásához.

5. Visszajelzések alapján történő iteráció: Folyamatosan kövesse nyomon az eszközök teljesítményét és gyűjtsön visszajelzéseket a felhasználóktól vagy az érintettektől. Használja ezeket a visszajelzéseket a fejlesztendő területek azonosítására és az eszköz definíciók iteratív finomítására. Például, ha a felhasználók jelzik, hogy az elemzés nem kezeli megfelelően a szarkazmust, hozzáadhat egy megjegyzést a leíráshoz:

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

Az eszköz definíciók iteratív finomításával a megfigyelt viselkedés és visszajelzések alapján fokozatosan javíthatja AI-vezérelt alkalmazásának teljesítményét és hatékonyságát. Ne feledje, hogy az eszköz definíciókat tartsa tisztán, tömören, és az adott feladatra összpontosítva. Rendszeresen tesztelje és validálja az eszközök közötti interakciókat, hogy megbizonyosodjon arról, összhangban vannak-e a kívánt eredményekkel.

Eszközök Összeállítása és Láncolása

Az eszközhasználat egyik legerősebb aspektusa, amelyre eddig csak utaltunk, az a képesség, hogy több eszközt összekapcsolhatunk és láncolhatunk komplex feladatok végrehajtásához. Az eszköz definíciók és azok be- és kimeneti formátumainak gondos tervezésével újrafelhasználható építőelemeket hozhat létre, amelyeket különböző módokon kombinálhat.

Vegyünk egy példát, ahol egy adatelemzési folyamatot épít az AI-vezérelt alkalmazásához. A következő eszközökkel rendelkezhet:

1. **`DataRetrieval`**: Egy eszköz, amely adatokat kér le egy adatbázisból vagy API-ból meghatározott kritériumok alapján.
2. **`DataProcessing`**: Egy eszköz, amely számításokat, átalakításokat vagy összesítéseket végez a lekért adatokon.
3. **`DataVisualization`**: Egy eszköz, amely a feldolgozott adatokat felhasználóbarát formátumban, például diagramok vagy grafikonok formájában jeleníti meg.

Ezeknek az eszközöknek az összekapcsolásával létrehozhat egy hatékony munkafolyamatot, amely lekéri a releváns adatokat, feldolgozza azokat, és értelmes módon mutatja be az eredményeket. Íme, hogyan nézhet ki az eszközhasználati munkafolyamat:

1. A nagy nyelvi modell fogad egy felhasználói kérést, amely egy adott termékkategória értékesítési adataival kapcsolatos betekintést kér.
2. A nagy nyelvi modell kiválasztja a `DataRetrieval` eszközt, és generálja a megfelelő bemeneti paramétereket a releváns értékesítési adatok adatbázisból történő lekéréséhez.
3. A lekért adatok "átkerülnek" a `DataProcessing` eszközhöz, amely kiszámítja olyan metrikákat, mint a teljes bevétel, átlagos eladási ár és növekedési ráta.
4. A feldolgozott adatokat ezután a `DataVisualization` eszköz dolgozza fel, amely vonzó diagramot vagy grafikont készít a betekintések ábrázolására, és visszaadja a diagram URL-jét a nagy nyelvi modellnek.
5. Végül a nagy nyelvi modell markdown használatával formázott választ generál a felhasználói kérésre, beépítve a vizualizált adatokat és összefoglalva a főbb megállapításokat.

Ezen eszközök összekapcsolásával létrehozhat egy zökkenőmentes adatelemzési munkafolyamatot, amely könnyen integrálható az alkalmazásba. Ennek a megközelítésnek a szépsége az, hogy minden eszköz függetlenül fejleszthető és tesztelhető, majd különböző módokon kombinálható különböző problémák megoldására.

Az eszközök zökkenőmentes összeállításának és láncolásának lehetővé tételéhez fontos, hogy minden eszközhöz világos be- és kimeneti formátumokat határozzunk meg.

Például a `DataRetrieval` eszköz elfogadhat olyan paramétereket, mint az adatbázis-kapcsolat részletei, táblanév és lekérdezési feltételek, és az eredményhalmazt strukturált JSON objektumként adhatja vissza. A `DataProcessing` eszköz ezután ezt a JSON objektumot várhatja bemenetként, és egy átalakított JSON objektumot állíthat elő kimenetként. Az eszközök közötti adatáramlás standardizálásával biztosíthatja a kompatibilitást és az újrafelhasználhatóságot.

Amikor az eszköz ökoszisztémáját tervezi, gondoljon arra, hogyan lehet különböző eszközöket kombinálni az alkalmazásában gyakori használati esetek kezelésére. Fontolja meg olyan magas szintű eszközök létrehozását, amelyek magukba foglalják a gyakori munkafolyamatokat vagy üzleti logikát, megkönnyítve a nagy nyelvi modell számára azok hatékony kiválasztását és használatát.

Ne feledje, az eszközhasználat ereje a rugalmasságban és modularitásban rejlik. A komplex feladatok kisebb, újrafelhasználható eszközökre bontásával robusztus és adaptálható AI-vezérelt alkalmazást hozhat létre, amely számos kihívással megbirkózhat.

Jövőbeli Irányok

Ahogy az AI-vezérelt alkalmazásfejlesztés területe fejlődik, további előrelépésekre számíthatunk az eszközhasználati képességekben. Néhány potenciális jövőbeli irány:

1. **Többlépcsős Eszközhasználat**: A nagy nyelvi modellek képesek lehetnek eldönteni, hányszor kell használniuk az eszközöket egy kielégítő válasz generálásához. Ez magában foglalhatja az eszközök többszöri kiválasztását és végrehajtását a közbenső eredmények alapján.
2. **Előre Definiált Eszközök**: Az AI platformok biztosíthatnak egy sor előre definiált eszközt, amelyeket a fejlesztők azonnal használhatnak, például Python értelmezőket, webes keresőeszközöket vagy általános segédfunkciókat.
3. **Zökkenőmentes Integráció**: Ahogy az eszközhasználat egyre elterjedtebbé válik, jobb integrációra számíthatunk az AI platformok és népszerű fejlesztési keretrendszerek között, megkönnyítve a fejlesztők számára az eszközhasználat beépítését alkalmazásaikba.

Az eszközhasználat egy hatékony technika, amely lehetővé teszi a fejlesztők számára, hogy teljes mértékben kihasználják a nagy nyelvi modellek potenciálját az AI-vezérelt alkalmazásokban. A nagy nyelvi modellek külső eszközökhöz és erőforrásokhoz való csatlakoztatásával dinamikusabb, intelligensebb és környezettudatosabb rendszereket hozhat létre, amelyek képesek alkalmazkodni a felhasználói igényekhez és értékes betekintéseket és műveleteket biztosítani.

Bár az eszközhasználat hatalmas lehetőségeket kínál, fontos tisztában lenni a potenciális kihívásokkal és megfontolásokkal. Az egyik kulcsfontosságú szempont az eszközök közötti interakciók összetettségének kezelése és a teljes rendszer stabilitásának és megbízhatóságának biztosítása. Kezelnie kell azokat a forgatókönyveket, ahol az eszközhívások meghiúsulhatnak, váratlan eredményeket adhatnak vissza, vagy teljesítménybeli következményekkel járhatnak. Emellett figyelembe kell vennie a biztonsági és hozzáférés-ellenőrzési intézkedéseket az eszközök jogosulatlan vagy rosszindulatú használatának megakadályozására. A megfelelő hibakezelési, naplózási és monitoring mechanizmusok kulcsfontosságúak az AI-vezérelt alkalmazás integritásának és teljesítményének fenntartásához.

Miközben saját projektjeiben feltárja az eszközhasználat lehetőségeit, ne feledje, hogy egyértelmű célokkal kezdjen, alakítson ki jól strukturált eszközdefiníciókat, és a visszajelzések és eredmények alapján iteráljon. A megfelelő megközelítéssel és szemlélettel az eszközhasználat új innovációs és értékteremtési szinteket nyithat meg MI-vezérelt alkalmazásaiban.

Adatfolyam-feldolgozás

Az adatok HTTP-n keresztüli streamelése, más néven szerver által küldött események (SSE), olyan mechanizmus, ahol a szerver folyamatosan küldi az adatokat a kliensnek, amint azok elérhetővé válnak, anélkül, hogy a kliensnek kifejezetten kérnie kellene azokat. Mivel a mesterséges intelligencia válasza fokozatosan generálódik, észszerű egy reszponzív felhasználói élményt biztosítani azáltal, hogy az AI kimenetét már a generálás közben megjelenítjük. És valójában minden általam ismert AI-szolgáltató API-ja kínál streaming válaszokat opcióként a kiegészítő végpontjaikon.

Ez a fejezet azért jelenik meg itt a könyvben, közvetlenül az Eszközök használata után, mert rendkívül hatékony lehet az eszközök használatának és a felhasználóknak szóló élő AI-válaszoknak az ötvözése. Ez lehetővé teszi olyan dinamikus és interaktív élmények létrehozását, ahol az AI feldolgozhatja a felhasználói bemeneteket, különböző eszközöket és funkciókat használhat saját belátása szerint, és valós idejű válaszokat adhat.

E zökkenőmentes interakció eléréséhez olyan adatfolyam-kezelőket kell írni, amelyek képesek továbbítani mind az AI által meghívott eszközfüggvény-hívásokat, mind az egyszerű szöveges kimenetet a végfelhasználónak. Az eszközfüggvény feldolgozása utáni ismételt végrehajtás szükségessége érdekes kihívást ad a feladatnak.

A ReplyStream implementálása

Az adatfolyam-feldolgozás megvalósításának bemutatására ez a fejezet részletesen megvizsgálja az Olympia-ban használt `ReplyStream` osztály egyszerűsített verzióját. Ennek az osztálynak a példányai átadhatók `stream` paraméterként olyan AI kliens könyvtárakban, mint a ruby-openai és az openrouter.

Íme, hogyan használom a `ReplyStream`-et az Olympia `PromptSubscriber`-ében, amely a Wisper segítségével figyeli az új felhasználói üzenetek létrehozását.

```
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...
```

A `ReplyStream` osztály a létrehozó prompt előfizetőre mutató `context` referencián kívül rendelkezik példányváltozókkal a fogadott adatok pufferelésére, valamint tömbökkel a adatfolyam-feldolgozás során meghívott függvénynevek és argumentumok nyilvántartására.

```
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
```

Az `initialize` metódus beállítja a `ReplyStream` példány kezdeti állapotát, inicializálja a puffert, a kontextust és egyéb változókat.

A `call` metódus a fő belépési pont az adatfolyam feldolgozásához. Ez a metódus egy `chunk` adatcsomagot (hash formájában) és egy opcionális `bytesize` paramétert fogad, amely a példánkban nincs használatban. Ezen belül az osztály mintaillesztést használ, hogy különböző forgatókönyveket kezeljen a beérkező adatcsomag szerkezete alapján.

A `deep_symbolize_keys` meghívása az adatcsomagra elegánsabbá teszi a mintaillesztést azáltal, hogy szimbólumokkal dolgozhatunk stringek helyett.

```
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
```

Az első minta, amit keresünk, egy eszközhívás a hozzá tartozó függvénynévvel együtt. Ha ilyet észlelünk, azt a `f_name` tömbbe helyezzük. A függvényneveket egy indexelt tömbben tároljuk, mivel a modell képes párhuzamos függvényhívásra, vagyis egyszerre több függvényt is végrehajtásra küldhet.

> A párhuzamos függvényhívás egy AI modell azon képessége, hogy több függvényhívást egyszerre hajtson végre, lehetővé téve, hogy ezeknek a függvényhívásoknak az eredményei párhuzamosan oldódjanak meg. Ez különösen hasznos, ha a függvények végrehajtása sok időt vesz igénybe, és csökkenti az API-val való oda-vissza utakat, ami jelentős token-megtakarítást eredményezhet.

Ezután meg kell keresnünk a függvényhívásokhoz tartozó argumentumokat.

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

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

A függvénynevekhez hasonlóan az argumentumokat is egy indexelt tömbbe helyezzük el.

A következő lépésben a normál, felhasználónak szánt üzeneteket figyeljük, amelyek a szerverről érkeznek, tokenenként, és a `new_content` változóban tárolódnak. Emellett szemmel kell tartanunk a `finish_reason` értékét is. Ez `nil` marad egészen a kimeneti szekvencia utolsó részletéig.

```
  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
```

Fontos megjegyezni, hogy mintaillesztési kifejezést adunk hozzá az MI-modell

szolgáltatótól érkező hibaüzenetek kezelésére. A helyi fejlesztői környezetekben kivételt dobunk, míg éles környezetben a hibát naplózzuk és lezárjuk a folyamatot.

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

A case utolsó else ága akkor fut le, ha egyik korábbi minta sem illeszkedett. Ez csupán egy biztonsági megoldás arra az esetre, ha az MI modell olyan felismerhetetlen adatcsomagokat kezdene küldeni, amelyekről így tudomást szerezhetünk.

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

A `send_to_client` metódus felelős a pufferelt tartalom klienshez való küldéséért. Ellenőrzi, hogy a puffer nem üres-e, frissíti a bot üzenet tartalmát, megjeleníti a bot üzenetet, és elmenti a tartalmat az adatbázisba az adatok megőrzésének biztosítása érdekében.

```
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
```

A `finalize` metódus akkor kerül meghívásra, amikor az adatfolyam-feldolgozás befejeződik. Végrehajtja a függvényhívásokat, amennyiben ilyenek érkeztek az adatfolyam során, frissíti a bot üzenetét a végleges tartalommal és egyéb releváns információkkal, valamint alaphelyzetbe állítja a függvényhívási előzményeket.

```
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
```

Ha a modell úgy dönt, hogy meghív egy függvényt, "továbbítanod" kell ezt a függvényhívást (név és argumentumok) olyan módon, hogy az végrehajtódjon, és a

`function_call` valamint `function_result` üzenetek hozzáadódjanak a beszélgetési jegyzőkönyvhöz

Tapasztalatom szerint jobb a függvényüzenetek létrehozását a kódbázis egy helyén kezelni, ahelyett, hogy az eszközimplementációkra hagyatkoznánk. Ez nem csak tisztább megoldás, de van egy nagyon fontos gyakorlati oka is: ha az MI modell meghív egy függvényt, és nem látja a hívás és eredmény üzeneteket a jegyzőkönyvben a következő körben, akkor *újra meg fogja hívni ugyanazt a függvényt.* Potenciálisan a végtelenségig. Ne feledd, hogy az MI teljesen állapotmentes, így hacsak nem tükrözöd vissza neki ezeket a függvényhívásokat, azok meg sem történtek számára.

```
# 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
```

A függvényhívási előzmények törlése a végrehajtás után ugyanolyan fontos, mint annak biztosítása, hogy a hívás és az eredmények bekerüljenek az átiratba, különben csak ugyanazokat a függvényeket hívnád meg újra és újra minden ciklusban.

A "Beszélgetési ciklus"

A `PromptSubscriber` osztályban a `PromptDeclarations` modul `prompt` metódusát használjuk a beszélgetési ciklus viselkedésének meghatározásához. Az `until` paraméter értéke `-> { bot_message.complete? }`, ami azt jelenti, hogy a ciklus addig folytatódik, amíg a `bot_message` nincs teljesként megjelölve.

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

De mikor jelöljük meg a `bot_message`-et befejezettként? Ha elfelejtetted, nézd meg újra a `finalize` metódus 13. sorát.

Tekintsük át a teljes adatfolyam-feldolgozási logikát.

1. A `PromptSubscriber` új felhasználói üzenetet kap a `message_created` metóduson keresztül, amelyet a Wisper publikálás/feliratkozás rendszer hív meg minden alkalommal, amikor a végfelhasználó új promptot hoz létre.
2. A `prompt` osztálymetódus deklaratív módon határozza meg a chat kiegészítési logika viselkedését a `PromptSubscriber` számára. Az AI modell végrehajtja a chat kiegészítést a felhasználó üzenetének tartalmával, egy új `ReplyStream` példánnyal mint stream paraméterrel, és a megadott ciklusfeltétellel.
3. Az AI modell feldolgozza a promptot és elkezdi generálni a választ. Ahogy a válasz streamelődik, a `ReplyStream` példány `call` metódusa minden adatdarabra meghívódik.
4. Ha az AI modell úgy dönt, hogy eszközfüggvényt hív meg, a függvény neve és argumentumai kivonásra kerülnek a darabból és tárolódnak az `f_name` és `f_arguments` tömbökben.
5. Ha az AI modell felhasználónak szánt tartalmat generál, az pufferelésre kerül és elküldődik a kliensnek a `send_to_client` metóduson keresztül.
6. Amint az adatfolyam feldolgozása befejeződik, a `finalize` metódus meghívódik. Ha bármilyen eszközfüggvény meghívásra került az adatfolyam során, azok a `PromptSubscriber` `dispatch` metódusával kerülnek végrehajtásra.
7. A `dispatch` metódus hozzáad egy `function_call` üzenetet a beszélgetési jegyzőkönyvhöz, végrehajtja a megfelelő eszközfüggvényt, és hozzáad egy `function_result` üzenetet a jegyzőkönyvhöz a függvényhívás eredményével.

8. Az eszközfüggvények végrehajtása után a függvényhívási előzmények törlődnek, hogy megakadályozzuk a duplikált függvényhívásokat a következő ciklusokban.
9. Ha nem került sor eszközfüggvény meghívására, a `finalize` metódus frissíti a `bot_message`-et a végleges tartalommal, megjelöli befejezettként, és elküldi a frissített üzenetet a kliensnek.
10. A ciklusfeltétel `-> { bot_message.complete? }` kiértékelődik. Ha a `bot_message` nincs befejezettként megjelölve, a ciklus folytatódik, és az eredeti prompt újra beküldésre kerül a frissített beszélgetési jegyzőkönyvvel.
11. A 3-10 lépések ismétlődnek, amíg a `bot_message` befejezettként nem jelölődik, jelezve, hogy az AI modell befejezte a válasz generálását és nincs szükség további eszközfüggvények végrehajtására.

Ennek a beszélgetési ciklusnak a megvalósításával lehetővé tesszük, hogy az AI modell oda-vissza interakcióba lépjen az alkalmazással, szükség szerint eszközfüggvényeket hajtson végre, és valós idejű válaszokat generáljon, amíg a beszélgetés természetes módon le nem zárul.

Az adatfolyam-feldolgozás és a beszélgetési ciklus kombinációja dinamikus és interaktív AI-vezérelt élményeket tesz lehetővé, ahol az AI modell feldolgozhatja a felhasználói bemeneteket, különböző eszközöket és függvényeket használhat, és valós idejű válaszokat adhat a fejlődő beszélgetési kontextus alapján.

Automatikus Folytatás

Fontos tisztában lenni az AI kimenet korlátaival. A legtöbb modellnek van egy maximális token száma, amit egyetlen válaszban generálhat, amit a `max_tokens` paraméter határoz meg. Ha az AI modell eléri ezt a limitet válaszgenerálás közben, hirtelen leáll és jelzi, hogy a kimenet csonkolva lett.

Az AI platform API-ból érkező streaming válaszban ezt a helyzetet úgy lehet észlelni, hogy megvizsgáljuk a `finish_reason` változót a darabban. Ha a `finish_reason`

értéke `"length"` (vagy valamilyen más, modellspecifikus kulcsérték), az azt jelenti, hogy a modell elérte a maximális token limitjét generálás közben, és a kimenet félbeszakadt.

Az egyik módja ennek a forgatókönyvnek a kecses kezelésére és a zökkenőmentes felhasználói élmény biztosítására az automatikus folytatási mechanizmus implementálása az adatfolyam-feldolgozási logikában. A hosszal kapcsolatos befejezési okok mintaillesztésének hozzáadásával választhatjuk azt, hogy ismételjük és automatikusan folytatjuk a kimenetet onnan, ahol abbamaradt.

Itt egy szándékosan egyszerűsített példa arra, hogyan módosíthatod a `call` metódust a `ReplyStream` osztályban az automatikus folytatás támogatásához:

```
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
```

```
27
28  def continue_cutoff
29    conversation.bot_message!(buffer.join, visible: false)
30    conversation.user_message!("please continue", visible: false)
31    bot_message.update_column(:created_at, Time.current)
32  end
```

Ebben a módosított verzióban, amikor a `finish_reason` csonkított kimenetet jelez, ahelyett, hogy véglegesítenénk az adatfolyamot, hozzáadunk egy üzenetpárt az átirathoz annak véglegesítése nélkül, az eredeti felhasználónak szánt válaszüzenetet az átirat "aljára" helyezzük a `created_at` attribútum frissítésével, majd hagyjuk, hogy a ciklus folytatódjon, így az MI ott folytatja a generálást, ahol abbahagyta.

Ne feledjük, hogy az MI kiegészítő végpont állapotmentes. Csak azt "tudja", amit az átiraton keresztül közlünk vele. Ebben az esetben úgy jelezzük az MI-nek, hogy megszakadt, hogy "láthatatlan" (a végfelhasználó számára nem látható) üzeneteket adunk az átirathoz. Ne feledjük azonban, hogy ez szándékosan egyszerűsített példa. Egy valódi implementációnak további átiratkezelést kellene végeznie annak biztosítására, hogy ne pazaroljunk tokeneket és/vagy ne zavarjuk össze az MI-t az átiratban megismételt asszisztensi üzenetekkel.

Az automatikus folytatás valódi implementációjának úgynevezett "megszakító logikát" is tartalmaznia kell a kontrollálatlan ciklusok megakadályozására. Ennek az az oka, hogy bizonyos típusú felhasználói promptok és alacsony `max_tokens` beállítások esetén az MI végtelenül folytathatná a felhasználói kimenet ciklikus generálását.

> Ne feledje, hogy minden ciklus külön kérést igényel, és minden kérés újra felhasználja a teljes átiratot. Mindenképpen mérlegelnie kell a felhasználói élmény és az API-használat közötti kompromisszumokat, amikor eldönti, hogy implementálja-e az automatikus folytatást az alkalmazásában. Az automatikus folytatás különösen veszélyesen költséges lehet, főleg prémium kereskedelmi modellek használata esetén.

Következtetés

Az adatfolyam-feldolgozás kritikus aspektusa az olyan MI-alapú alkalmazások fejlesztésének, amelyek eszközhasználatot kombinálnak élő MI-válaszokkal. Az MI platform API-k által közvetített adatfolyam hatékony kezelésével zökkenőmentes és interaktív felhasználói élményt biztosíthat, kezelni tudja a nagy válaszokat, optimalizálhatja az erőforrás-használatot és elegánsan kezelheti a hibákat.

A bemutatott `Conversation::ReplyStream` osztály szemlélteti, hogyan lehet az adatfolyam-feldolgozást implementálni egy Ruby alkalmazásban mintaillesztés és eseményvezérelt architektúra használatával. Az adatfolyam-feldolgozási technikák megértésével és kihasználásával felszabadíthatja az MI-integráció teljes potenciálját az alkalmazásaiban, és hatékony, lebilincselő felhasználói élményt nyújthat.

Öngyógyító adatok

Az öngyógyító adatok egy hatékony megközelítés az adatok integritásának, konzisztenciájának és minőségének biztosítására az alkalmazásokban, kihasználva a nagy nyelvi modellek (LLM-ek) képességeit. A minták ezen kategóriája arra az elképzelésre összpontosít, hogy a mesterséges intelligencia segítségével automatikusan észleljük, diagnosztizáljuk és javítsuk az adatrendellenességeket, következetlenségeket vagy hibákat, ezáltal csökkentve a fejlesztőkre nehezedő terheket és fenntartva az adatok megbízhatóságának magas szintjét.

Lényegében az öngyógyító adat minták felismerik, hogy az adat minden alkalmazás életereje, és pontosságának és integritásának biztosítása kulcsfontosságú az alkalmazás megfelelő működéséhez és felhasználói élményéhez. Az adatminőség kezelése és fenntartása azonban összetett és időigényes feladat lehet, különösen ahogy az alkalmazások mérete és összetettsége növekszik. Itt jön képbe a mesterséges intelligencia ereje.

Az öngyógyító adat mintákban MI munkavégzőket alkalmaznak az alkalmazás adatainak folyamatos figyelésére és elemzésére. Ezek a modellek képesek megérteni és értelmezni az adatok mintázatait, összefüggéseit és rendellenességeit. Természetes nyelvfeldolgozási és megértési képességeiket kihasználva azonosíthatják az adatokban rejlő potenciális problémákat vagy következetlenségeket, és megfelelő lépéseket tehetnek azok kijavítására.

Az öngyógyító adatok folyamata általában több kulcsfontosságú lépést foglal magában:

1. **Adatfelügyelet**: Az MI munkavégzők folyamatosan figyelik az alkalmazás adatfolyamait, adatbázisait vagy tárolórendszereit, keresve a rendellenességek, következetlenségek vagy hibák jeleit. Alternatív megoldásként aktiválhat egy MI komponenst egy kivétel reakciójaként.
2. **Anomália észlelés**: Amikor egy problémát észlelnek, az MI munkavégző részletesen elemzi az adatokat, hogy azonosítsa a probléma pontos természetét és hatókörét. Ez magában foglalhatja hiányzó értékek, következetlen formátumok vagy az előre meghatározott szabályokat vagy korlátozásokat megsértő adatok észlelését.
3. **Diagnózis és javítás**: Miután azonosította a problémát, az MI munkavégző az adatterület ismereteit és megértését használja fel a megfelelő cselekvési irány meghatározásához. Ez magában foglalhatja az adatok automatikus javítását, hiányzó értékek kitöltését, vagy szükség esetén a probléma jelzését emberi beavatkozáshoz.
4. **Folyamatos tanulás (opcionális, használati esettől függően)**: Ahogy az MI munkavégző különböző adatproblémákkal találkozik és old meg, kimenetet generálhat arról, hogy mi történt és hogyan reagált. Ez a metaadat olyan tanulási folyamatokba táplálható, amelyek lehetővé teszik az Ön (és esetleg a mögöttes modell, finomhangolás révén) számára, hogy idővel hatékonyabbá és eredményesebbé váljon az adatrendellenességek azonosításában és megoldásában.

Az adatproblémák automatikus észlelésével és javításával biztosíthatja, hogy

alkalmazása magas minőségű, megbízható adatokkal működjön. Ez csökkenti a hibák, következetlenségek vagy adattal kapcsolatos hibák kockázatát, amelyek befolyásolhatnák az alkalmazás funkcionalitását vagy felhasználói élményét.

Amint MI munkavégzők kezelik az adatfelügyelet és -javítás feladatát, erőfeszítéseit az alkalmazás más kritikus aspektusaira összpontosíthatja. Ez időt és erőforrásokat takarít meg, amelyeket egyébként kézi adattisztításra és karbantartásra kellene fordítani. Valójában, ahogy alkalmazásai mérete és összetettsége növekszik, az adatminőség kézi kezelése egyre nagyobb kihívást jelent. Az "Öngyógyító adatok" minták hatékonyan skálázódnak a mesterséges intelligencia erejét kihasználva nagy adatmennyiségek kezelésére és a problémák valós idejű észlelésére.

Természetükből adódóan az MI modellek kevés vagy semmilyen felügyelet nélkül képesek alkalmazkodni a változó adatmintákhoz, sémákhoz vagy követelményekhez. Amíg az utasításaik megfelelő iránymutatást nyújtanak, különösen a kívánt eredményekre vonatkozóan, alkalmazása képes lehet fejlődni és új adatforgatókönyveket kezelni kiterjedt kézi beavatkozás vagy kódváltoztatások nélkül.

Az öngyógyító adat minták jól illeszkednek a többi mintakategóriához, amelyekről beszéltünk, mint például a "Munkavégzők sokasága". Az öngyógyító adat képesség egy speciális munkavégző típusnak tekinthető, amely kifejezetten az adatminőség és integritás biztosítására összpontosít. Ez a fajta munkavégző más MI munkavégzőkkel együtt működik, mindegyik hozzájárulva az alkalmazás funkcionalitásának különböző aspektusaihoz.

Az öngyógyító adat minták gyakorlati megvalósítása gondos tervezést és az MI modellek integrálását igényli az alkalmazás architektúrájába. Az adatvesztés és -sérülés kockázata miatt egyértelmű irányelveket kell meghatároznia arról, hogyan fogja használni ezt a technikát. Figyelembe kell vennie olyan tényezőket is, mint a teljesítmény, skálázhatóság és adatbiztonság.

Gyakorlati esettanulmány: Hibás JSON javítása

Az öngyógyító adatok egyik legpraktikusabb és legkényelmesebb felhasználási módja egyben nagyon egyszerűen elmagyarázható: a hibás JSON javítása.

Ez a technika alkalmazható az LLM-ek által generált tökéletlen vagy következetlen adatok, például a hibás JSON kezelésének gyakori kihívására, és megközelítést kínál ezek automatikus észlelésére és javítására.

Az Olympiánál rendszeresen találkozom olyan esetekkel, amikor az LLM-ek nem tökéletesen érvényes JSON adatokat generálnak. Ez különböző okokból fordulhat elő, például amikor az LLM megjegyzéseket fűz a tényleges JSON kód elé vagy után, vagy amikor szintaktikai hibákat vezet be, mint például hiányzó vesszők vagy nem escapelt idézőjelek. Ezek a problémák elemzési hibákhoz vezethetnek és megzavarhatják az alkalmazás működését.

A probléma megoldására egy praktikus megoldást dolgoztam ki egy JsonFixer osztály formájában. Ez az osztály az "Öngyógyító Adat" mintát testesíti meg azáltal, hogy a hibás JSON-t bemenetként veszi, és egy LLM segítségével kijavítja azt, miközben a lehető legtöbb információt és szándékot megőrzi.

```
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
```

Figyeld meg, hogy a `JsonFixer` hogyan használja a Ventriloquist eszközt az AI válaszainak irányítására.

A JSON adatok önjavító folyamata a következőképpen működik:

1. **JSON generálás**: Egy LLM-et használunk JSON adatok generálására meghatározott promptok vagy követelmények alapján. Azonban az LLM-ek természetéből adódóan a generált JSON nem mindig tökéletesen érvényes. A JSON elemző természetesen `ParserError` hibát fog dobni, ha érvénytelen JSON-t kap.

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

Vegye figyelembe, hogy a kivétel üzenet szintén átadásra kerül a `JSONFixer` hívásnak, így nem kell teljesen feltételeznie, hogy mi a probléma az adatokkal, különösen mivel az elemző gyakran pontosan megmondja, hogy mi a hiba.

2. **LLM-alapú Javítás**: A `JSONFixer` osztály visszaküldi a hibás JSON-t egy LLM-nek, egy specifikus prompttal vagy utasítással együtt, hogy javítsa ki a JSON-t, miközben a lehető legnagyobb mértékben megőrzi az eredeti információt és

szándékot. Az LLM, amely hatalmas mennyiségű adaton lett betanítva és érti a JSON szintaxist, megpróbálja kijavítani a hibákat és érvényes JSON karakterláncot generálni. A Válasz Körülhatárolás szolgál az LLM kimenetének korlátozására, és a Mixtral 8x7B-t választottuk AI modellként, mivel különösen alkalmas az ilyen típusú feladatokra.

3. **Validálás és Integráció**: Az LLM által visszaadott javított JSON karakterláncot maga a `JSONFixer` osztály elemzi, mivel a `chat_completion(json: true)` hívást használta. Ha a javított JSON átmegy a validáláson, integrálódik az alkalmazás munkafolyamatába, lehetővé téve az alkalmazás számára az adatok zökkenőmentes feldolgozásának folytatását. A hibás JSON "meggyógyult".

Bár többször is megírtam és újraírtam a saját `JSONFixer` implementációmat, kétlem, hogy az összes verzióra fordított idő több lenne egy-két óránál.

Fontos megjegyezni, hogy a szándék megőrzése kulcsfontosságú eleme bármely öngyógyító adat mintának. Az LLM-alapú javítási folyamat célja az eredeti információ és szándék lehető legnagyobb mértékű megőrzése. Ez biztosítja, hogy a javított JSON megtartsa szemantikai jelentését és hatékonyan használható legyen az alkalmazás kontextusában.

Az "Öngyógyító Adat" megközelítés ezen gyakorlati implementációja az Olympiában világosan demonstrálja, hogyan lehet az AI-t, különösen az LLM-eket felhasználni valós adatkezelési kihívások megoldására. Bemutatja a hagyományos programozási technikák és az AI képességek kombinálásának erejét robusztus és hatékony alkalmazások építésében.

Postel törvénye és az "Öngyógyító Adat" minta

Az "Öngyógyító Adat", ahogy azt a JSONFixer osztály példázza, jól illeszkedik a

Postel-törvényként, vagy más néven Robusztussági Elvként ismert alapelvhez. Postel törvénye így szól:

"Légy konzervatív abban, amit teszel, és liberális abban, amit másoktól elfogadsz."

Ez az elv, amelyet eredetileg Jon Postel, az internet korai úttörője fogalmazott meg, hangsúlyozza annak fontosságát, hogy olyan rendszereket építsünk, amelyek toleránsak a különböző vagy akár enyhén hibás bemenetekkel szemben, miközben szigorúan betartják a meghatározott protokollokat a kimenetek küldésekor.

Az "Öngyógyító Adat" kontextusában a JSONFixer osztály megtestesíti Postel törvényét azzal, hogy liberálisan fogadja el az LLM-ek által generált hibás vagy tökéletlen JSON adatokat. Nem utasítja el azonnal és nem hibázik, amikor olyan JSON-nal találkozik, amely nem szigorúan követi az elvárt formátumot. Ehelyett toleráns megközelítést alkalmaz, és megpróbálja kijavítani a JSON-t az LLM-ek erejét felhasználva.

Azzal, hogy liberálisan fogadja el a tökéletlen JSON-t, a JSONFixer osztály robusztusságot és rugalmasságot mutat. Elismeri, hogy a valós világban az adatok gyakran különböző formákban érkeznek, és nem mindig felelnek meg a szigorú előírásoknak. Ezen eltérések kecses kezelésével és javításával az osztály biztosítja, hogy az alkalmazás zökkenőmentesen működhessen tovább, még tökéletlen adatok jelenlétében is.

Másrészt a JSONFixer osztály követi Postel törvényének konzervatív aspektusát is a kimenet tekintetében. Miután az LLM-ek segítségével kijavította a JSON-t, az osztály ellenőrzi a javított JSON-t, hogy szigorúan megfeleljen az elvárt formátumnak. Fenntartja az adatok integritását és helyességét, mielőtt továbbítaná azokat az alkalmazás más részeibe. Ez a konzervatív megközelítés garantálja, hogy a JSONFixer osztály kimenete megbízható és konzisztens, elősegítve az együttműködési képességet és megakadályozva a hibák terjedését.

Érdekes tények Jon Postelről:

- Jon Postel (1943-1998) amerikai számítógéptudós volt, aki kulcsszerepet játszott az Internet fejlesztésében. Az "Internet Isteneként" ismerték jelentős hozzájárulásai miatt az alapvető protokollokhoz és szabványokhoz.
- Postel volt a Request for Comments (RFC) dokumentumsorozat szerkesztője, amely az Internetről szóló technikai és szervezeti jegyzetek sorozata. Több mint 200 RFC szerzője vagy társszerzője volt, beleértve az olyan alapvető protokollokat, mint a TCP, IP és SMTP.
- Technikai hozzájárulásain túl Postel szerény és együttműködő megközelítéséről volt ismert. Hitt a konszenzus elérésének és az együttműködés fontosságában egy robusztus és interoperábilis hálózat kiépítése érdekében.
- Postel a University of Southern California (USC) Information Sciences Institute (ISI) Számítógépes Hálózatok Részlegének igazgatójaként dolgozott 1977-től korai haláláig, 1998-ig.
- Hatalmas hozzájárulásainak elismeréseként Postelt posztumusz a rangos Turing-díjjal tüntették ki 1998-ban, amelyet gyakran a "Számítástechnika Nobel-díjaként" emlegetnek.

A `JSONFixer` osztály a robusztusságot, a rugalmasságot és az interoperabilitást támogatja, amelyek Postel karrierje során végig alapvető értékek voltak. Olyan rendszerek építésével, amelyek tolerálják a tökéletlenségeket, miközben szigorúan betartják a protokollokat, olyan alkalmazásokat hozhatunk létre, amelyek ellenállóbbak és alkalmazkodóképesebbek a valós kihívásokkal szemben.

Megfontolások és Ellenjavallatok

Az önjavító adatmegközelítések alkalmazhatósága teljes mértékben attól függ, hogy milyen típusú adatokat kezel az alkalmazásunk. Van oka annak, hogy miért nem érdemes egyszerűen monkeypatch-elni a `JSON.parse` függvényt, hogy automatikusan

javítsa ki az *összes JSON elemzési hibát* az alkalmazásban: nem minden hiba javítható vagy javítandó automatikusan.

Az önjavítás különösen problémás, amikor az adatkezelésre és -feldolgozásra vonatkozó szabályozási vagy megfelelőségi követelményekkel párosul. Egyes iparágakban, mint például az egészségügyben és a pénzügyi szektorban, olyan szigorú szabályozások vonatkoznak az adatok integritására és auditálhatóságára, hogy bármilyen "fekete doboz" jellegű adatjavítás megfelelő felügyelet vagy naplózás nélkül megsértheti ezeket a szabályozásokat. Kulcsfontosságú annak biztosítása, hogy az általunk kidolgozott önjavító adattechnikák összhangban legyenek az alkalmazandó jogi és szabályozási keretrendszerekkel.

Az önjavító adattechnikák alkalmazása, különösen a mesterséges intelligencia modelleket alkalmazó megoldások, jelentős hatással lehetnek az alkalmazás teljesítményére és erőforrás-felhasználására. A nagy mennyiségű adat feldolgozása AI modelleken keresztül a hibák észlelése és javítása céljából számítási szempontból intenzív lehet. Fontos mérlegelni az önjavító adatok előnyei és a kapcsolódó teljesítmény- és erőforrásköltségek közötti kompromisszumokat.

Mindezek után nézzük meg részletesen, hogy milyen tényezőket kell figyelembe venni ennek a hatékony megközelítésnek az alkalmazásakor.

Adatkritikusság

Az önjavító adattechnikák alkalmazásának mérlegelésekor kulcsfontosságú a feldolgozott adatok kritikusságának értékelése. A kritikusság szintje az adatok fontosságára és érzékenységére utal az alkalmazás és az üzleti terület kontextusában.

Bizonyos esetekben az adathibák automatikus javítása nem megfelelő, különösen ha az adatok nagyon érzékenyek vagy jogi következményekkel járnak. Például vegyük a következő eseteket:

1. **Pénzügyi Tranzakciók**: A pénzügyi alkalmazásokban, például banki

rendszerekben vagy kereskedési platformokon, az adatok pontossága kiemelkedő fontosságú. Még a kisebb hibák is jelentős következményekkel járhatnak a pénzügyi adatokban, például helytelen számlaegyenlegek, rosszul irányított pénzeszközök vagy hibás kereskedési döntések formájában. Ezekben az esetekben az automatizált javítások alapos ellenőrzés és auditálás nélkül elfogadhatatlan kockázatokat hordozhatnak.

2. **Egészségügyi Nyilvántartások**: Az egészségügyi alkalmazások rendkívül érzékeny és bizalmas betegadatokkal dolgoznak. Az egészségügyi nyilvántartásokban lévő pontatlanságok súlyos következményekkel járhatnak a betegbiztonságra és a kezelési döntésekre nézve. Az egészségügyi adatok automatikus módosítása megfelelő szakmai felügyelet és képesített egészségügyi szakemberek általi validálás nélkül megsértheti a szabályozási követelményeket és veszélyeztetheti a betegek jólétét.
3. **Jogi Dokumentumok**: A jogi dokumentumokat, például szerződéseket, megállapodásokat vagy bírósági beadványokat kezelő alkalmazások szigorú pontosságot és integritást igényelnek. Még a kisebb hibák is jelentős jogi következményekkel járhatnak. Az automatizált javítások ezen a területen nem feltétlenül megfelelőek, mivel az adatok gyakran jogi szakértők általi manuális áttekintést és ellenőrzést igényelnek az érvényesség és végrehajthatóság biztosítása érdekében.

Ezekben a kritikus adatkezelési forgatókönyvekben az automatizált javításokkal járó kockázatok gyakran meghaladják a potenciális előnyöket. A hibák bevezetésének vagy az adatok helytelen módosításának következményei súlyosak lehetnek, pénzügyi veszteségekhez, jogi felelősséghez, vagy akár személyi sérülésekhez vezethetnek.

Kritikus adatok kezelésekor elengedhetetlen a manuális ellenőrzési és validálási folyamatok előtérbe helyezése. Az emberi felügyelet és szakértelem kulcsfontosságú az adatok pontosságának és integritásának biztosításában. Az automatizált önjavító technikák továbbra is alkalmazhatók a potenciális hibák vagy következetlenségek

jelzésére, de a végső javítási döntéseknek emberi megítélést és jóváhagyást kell tartalmazniuk.

Fontos azonban megjegyezni, hogy nem minden adat rendelkezik ugyanazzal a kritikussági szinttel egy alkalmazáson belül. Ugyanazon alkalmazáson belül lehetnek olyan adathalmazok, amelyek kevésbé érzékenyek vagy kisebb hatással járnak a hibák esetén. Ilyen esetekben az önjavító adattechnikák szelektíven alkalmazhatók ezekre a specifikus adathalmazokra, míg a kritikus adatok továbbra is manuális ellenőrzés alatt maradnak.

A kulcs az, hogy gondosan értékeljük az alkalmazásban található egyes adatkategóriák kritikusságát, és világos irányelveket és folyamatokat határozzunk meg a javítások kezelésére a kapcsolódó kockázatok és következmények alapján. A kritikus (pl. főkönyvek, egészségügyi nyilvántartások) és nem kritikus adatok (pl. levelezési címek, erőforrás-figyelmeztetések) megkülönböztetésével egyensúlyt teremthetünk az önjavító adattechnikák előnyeinek kihasználása és a szigorú ellenőrzés és felügyelet fenntartása között.

Végső soron a döntést az önjavító adattechnikák kritikus adatokra való alkalmazásáról a szakterületi szakértőkkel, jogi tanácsadókkal és más érintett érdekeltekkel konzultálva kell meghozni. Elengedhetetlen az alkalmazás adataival kapcsolatos specifikus követelmények, szabályozások és kockázatok figyelembevétele, és az adatjavítási stratégiák ennek megfelelő összehangolása.

Hibák Súlyossága

Az önjavító adattechnikák alkalmazásakor fontos értékelni az adathibák súlyosságát és hatását. Nem minden hiba egyenértékű, és a megfelelő eljárás változhat a probléma súlyosságától függően.

A kisebb következetlenségek vagy formázási problémák alkalmasak lehetnek automatikus javításra. Például egy törött JSON javítására szolgáló önjavító

adatfeldolgozó képes kezelni a hiányzó vesszőket vagy a nem escapelt idézőjeleket anélkül, hogy jelentősen megváltoztatná az adatok jelentését vagy szerkezetét. Az ilyen típusú hibák gyakran egyszerűen javíthatók és minimális hatással vannak az általános adatintegritásra.

Azonban a súlyosabb hibák esetében, amelyek alapvetően megváltoztatják az adatok jelentését vagy integritását, más megközelítésre lehet szükség. Ilyen esetekben az automatizált javítások nem feltétlenül elegendőek, és emberi beavatkozásra lehet szükség az adatok pontosságának és érvényességének biztosításához.

Itt jön képbe az a koncepció, hogy magát a mesterséges intelligenciát használjuk a hibák súlyosságának meghatározásához. A mesterséges intelligencia modellek képességeit kihasználva olyan öngyógyító adatfeldolgozókat tervezhetünk, amelyek nem csak javítják a hibákat, hanem értékelik is azok súlyosságát, és megalapozott döntéseket hoznak a kezelésükkel kapcsolatban.

Vegyünk például egy öngyógyító adatfeldolgozót, amely az ügyfél-adatbázisba áramló adatok következetlenségeinek javításáért felelős. Az adatfeldolgozó úgy tervezhető, hogy elemezze az adatokat és azonosítsa a potenciális hibákat, például a hiányzó vagy ellentmondásos információkat. Azonban ahelyett, hogy automatikusan javítana minden hibát, az adatfeldolgozó további eszközhívásokkal látható el, amelyek lehetővé teszik számára, hogy a súlyos hibákat emberi felülvizsgálatra jelölje.

Íme egy példa arra, hogyan lehet ezt megvalósítani:

```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
```

Ebben a példában a `CustomerDataHealer` worker az ügyféladatokban található következetlenségek azonosítására és javítására szolgál. Ismét a válaszhatárolást és a hasbeszélőt használjuk a strukturált kimenet érdekében. Fontos megemlíteni, hogy a

worker rendszerutasításai tartalmazzák a `flag_for_review` függvény használatát súlyos hibák esetén.

Amikor a worker feldolgozza az ügyféladatokat, elemzi azokat és megpróbálja kijavítani az esetleges következetlenségeket. Ha a worker úgy ítéli meg, hogy a hibák súlyosak és emberi beavatkozást igényelnek, használhatja a `flag_for_review` eszközt az adatok megjelölésére, megadva a jelölés okát.

A `chat_completion` metódust a `json: true` paraméterrel hívjuk meg, hogy a javított ügyféladatokat JSON-ként elemezze. Nincs lehetőség a függvényhívás utáni ismétlésre, így az eredmény üres lesz, ha a `flag_for_review` meghívásra került. Ellenkező esetben az ügyfél adatai frissülnek az áttekintett és esetlegesen javított adatokkal.

A hibasúlyosság értékelésének és az emberi felülvizsgálatra való jelölés lehetőségének beépítésével az öngyógyító adatfeldolgozó worker intelligensebbé és alkalmazkodóképesebbé válik. Automatikusan kezelheti a kisebb hibákat, míg a súlyos hibákat emberi szakértőkhöz továbbíthatja kézi beavatkozásra.

A hibasúlyosság meghatározásának konkrét kritériumai a worker utasításaiban definiálhatók a szakterületi tudás és az üzleti követelmények alapján. A súlyosság értékelésekor olyan tényezőket lehet figyelembe venni, mint az adatok integritására gyakorolt hatás, az adatvesztés vagy -sérülés lehetősége, valamint a helytelen adatok következményei.

A mesterséges intelligencia felhasználásával a hibasúlyosság értékelésére és az emberi beavatkozás lehetőségének biztosításával az öngyógyító adattechnikák egyensúlyt teremthetnek az automatizálás és az adatpontosság fenntartása között. Ez a megközelítés biztosítja, hogy a kisebb hibák hatékonyan javításra kerüljenek, míg a súlyos hibák megkapják a szükséges figyelmet és szakértelmet az emberi felülvizsgálóktól.

Tartományi komplexitás

Az öngyógyító adattechnikák alkalmazásának mérlegelésekor fontos értékelni az adattartomány komplexitását és a szerkezetét, valamint kapcsolatait szabályozó szabályokat. A tartomány összetettsége jelentősen befolyásolhatja az automatizált adatjavítási megközelítések hatékonyságát és megvalósíthatóságát.

Az öngyógyító adattechnikák jól működnek, amikor az adatok jól meghatározott mintákat és korlátozásokat követnek. Olyan tartományokban, ahol az adatszerkezet viszonylag egyszerű, és az adatelemek közötti kapcsolatok egyértelműek, az automatizált javítások nagy megbízhatósággal alkalmazhatók. Például a formázási problémák javítása vagy az alapvető adattípus-korlátozások érvényesítése gyakran hatékonyan kezelhető öngyógyító adatworkerekkel.

Azonban ahogy az adattartomány komplexitása növekszik, az automatizált adatjavítással kapcsolatos kihívások is nőnek. Az összetett üzleti logikával, bonyolult kapcsolatokkal az adategységek között, vagy tartományspecifikus szabályokkal és kivételekkel rendelkező területeken az öngyógyító adattechnikák nem mindig ragadják meg a finomságokat, és nem kívánt következményeket okozhatnak.

Vegyünk egy példát egy összetett tartományra: egy pénzügyi kereskedési rendszert. Ebben a tartományban az adatok különböző pénzügyi eszközöket, piaci adatokat, kereskedési szabályokat és szabályozási követelményeket tartalmaznak. A különböző adatelemek közötti kapcsolatok összetettek lehetnek, és az adatok érvényességét és konzisztenciáját szabályozó szabályok nagymértékben tartományspecifikusak lehetnek.

Egy ilyen összetett tartományban egy öngyógyító adatworkernek, amely a kereskedési adatok következetlenségeinek javításával foglalkozik, mély megértéssel kell rendelkeznie a tartományspecifikus szabályokról és korlátozásokról. Figyelembe kell vennie olyan tényezőket, mint a piaci szabályozások, kereskedési limitek, kockázatszámítások és elszámolási eljárások. Az automatizált javítások ebben a kontextusban nem mindig ragadják meg a tartomány teljes összetettségét, és

akaratlanul is hibákat vezethetnek be vagy megsérthetik a tartományspecifikus szabályokat.

A tartományi komplexitás kihívásainak kezelésére az öngyógyító adattechnikák fejleszthetők a tartományspecifikus tudás és szabályok beépítésével a mesterséges intelligencia modellekbe és workerekbe. Ez a következő technikákkal érhető el:

1. **Tartományspecifikus betanítás**: Az öngyógyító adatokhoz használt MI-modellek irányíthatók vagy akár finomhangolhatók tartományspecifikus adatkészleteken, amelyek megragadják az adott tartomány összetettségét és szabályait. A modellek reprezentatív adatoknak és forgatókönyveknek való kitettségével megtanulhatják a tartományra jellemző mintákat, korlátozásokat és kivételeket.
2. **Szabályalapú korlátozások**: Az öngyógyító adatworkerek kiegészíthetők explicit szabályalapú korlátozásokkal, amelyek tartományspecifikus tudást kódolnak. Ezeket a szabályokat szakterületi szakértők határozhatják meg és integrálhatják az adatjavítási folyamatba. Az MI-modellek ezután felhasználhatják ezeket a szabályokat döntéseik irányításához és a tartományspecifikus követelményeknek való megfelelés biztosításához.
3. **Együttműködés szakterületi szakértőkkel**: Összetett tartományokban kulcsfontosságú a szakterületi szakértők bevonása az öngyógyító adattechnikák tervezésébe és fejlesztésébe. A szakterületi szakértők értékes betekintést nyújthatnak az adatok összetettségébe, az üzleti szabályokba és a lehetséges határesetekbe. Tudásuk beépíthető az MI-modellekbe és workerekbe az automatizált adatjavítások pontosságának és megbízhatóságának javítása érdekében az ember a hurokban minták használatával.
4. **Fokozatos és iteratív megközelítés**: Összetett tartományok kezelésekor gyakran előnyös az öngyógyító adatok fokozatos és iteratív megközelítése. Ahelyett, hogy megpróbálnánk automatizálni a javításokat a teljes tartományra egyszerre, összpontosítsunk olyan specifikus altartományokra vagy adatkategóriákra, ahol

a szabályok és korlátozások jól érthetők. Fokozatosan bővítsük az öngyógyító technikák hatókörét, ahogy a tartomány megértése növekszik és a technikák hatékonynak bizonyulnak.

Az adattartomány összetettségének figyelembevételével és a szakterület-specifikus ismeretek öngyógyító adattechnikákba való beépítésével egyensúlyt teremthetünk az automatizálás és a pontosság között. Fontos felismerni, hogy az öngyógyító adatok nem jelentenek univerzális megoldást, és a megközelítést minden szakterület egyedi követelményeihez és kihívásaihoz kell igazítani.

Összetett területeken a leghatékonyabb lehet egy olyan hibrid megközelítés, amely ötvözi az öngyógyító adattechnikákat az emberi szakértelemmel és felügyelettel. Az automatizált javítások kezelhetik a rutinszerű és jól meghatározott eseteket, míg a komplex forgatókönyveket vagy kivételeket emberi áttekintésre és beavatkozásra lehet jelölni. Ez az együttműködésen alapuló megközelítés biztosítja, hogy az automatizálás előnyei megvalósuljanak, miközben fenntartja a szükséges ellenőrzést és pontosságot az összetett adatterületeken.

Megmagyarázhatóság és átláthatóság

A megmagyarázhatóság az AI modellek döntései mögötti érvelés megértésének és értelmezésének képességére utal, míg az átláthatóság az adatjavítási folyamat világos láthatóságának biztosítását jelenti.

Sok összefüggésben az adatmódosításoknak ellenőrizhetőnek és indokolhatónak kell lenniük. Az érdekelt felek, beleértve az üzleti felhasználókat, ellenőröket és szabályozó testületeket, magyarázatot kérhetnek arra, hogy miért történtek bizonyos adatjavítások, és hogyan jutottak az AI modellek ezekre a döntésekre. Ez különösen fontos olyan területeken, ahol az adatok pontosságának és integritásának jelentős következményei vannak, például a pénzügy, az egészségügy és a jogi ügyek területén.

A megmagyarázhatóság és átláthatóság igényének kielégítése érdekében az öngyógyító adattechnikáknak olyan mechanizmusokat kell tartalmazniuk, amelyek betekintést

nyújtanak az AI modellek döntéshozatali folyamatába. Ez különböző megközelítésekkel érhető el:

1. **Gondolatmenet**: Ha megkérjük a modellt, hogy "hangosan" magyarázza el gondolkodását az adatok módosítása előtt, az megkönnyítheti a döntéshozatali folyamat megértését, és emberi nyelven érthető magyarázatokat generálhat a végrehajtott javításokról. A kompromisszum egy kicsit több összetettség a magyarázat és a strukturált adatkimenet szétválasztásában, ami kezelhető...
2. **Magyarázat generálása**: Az öngyógyító adatfeldolgozókat fel lehet szerelni azzal a képességgel, hogy emberi nyelven érthető magyarázatokat generáljanak az általuk végzett javításokról. Ez elérhető úgy, hogy a modellt arra kérjük, hogy a döntéshozatali folyamatát *magukba az adatokba integrált* könnyen érthető magyarázatok formájában adja ki. Például egy öngyógyító adatfeldolgozó készíthet olyan jelentést, amely kiemeli az azonosított adatkövetkezetlenségeket, az alkalmazott javításokat és az ezek mögötti indoklást.
3. **Jellemzők fontossága**: Az AI modellek utasíthatók az adatjavítási folyamatban szereplő különböző jellemzők vagy tulajdonságok fontosságára vonatkozó információkkal az utasításaik részeként. Ezek az utasítások pedig láthatóvá tehetők az emberi érdekelt felek számára. A modell döntéseit befolyásoló kulcsfontosságú tényezők azonosításával az érdekelt felek betekintést nyerhetnek a javítások mögötti érvelésbe és értékelhetik azok érvényességét.
4. **Naplózás és ellenőrzés**: Az átfogó naplózási és ellenőrzési mechanizmusok megvalósítása kulcsfontosságú az öngyógyító adatfolyamat átláthatóságának fenntartásához. Az AI modellek által végzett minden adatjavítást naplózni kell, beleértve az eredeti adatokat, a javított adatokat és a végrehajtott konkrét műveleteket. Ez az ellenőrzési nyomvonal lehetővé teszi a visszamenőleges elemzést és világos nyilvántartást biztosít az adatokon végzett módosításokról.
5. **Ember a folyamatban megközelítés**: Az ember a folyamatban megközelítés beépítése növelheti az öngyógyító adattechnikák megmagyarázhatóságát és átláthatóságát. Az emberi szakértők bevonásával az AI által generált javítások

áttekintésébe és validálásába a szervezetek biztosíthatják, hogy a javítások összhangban legyenek a szakterületi ismeretekkel és az üzleti követelményekkel. Az emberi felügyelet további elszámoltathatósági szintet ad, és lehetővé teszi az AI modellekben esetlegesen előforduló torzítások vagy hibák azonosítását.

6. **Folyamatos monitoring és értékelés**: Az öngyógyító adattechnikák teljesítményének rendszeres figyelemmel kísérése és értékelése elengedhetetlen az átláthatóság és a bizalom fenntartásához. Az AI modellek pontosságának és hatékonyságának időbeli értékelésével a szervezetek azonosíthatják az esetleges eltéréseket vagy rendellenességeket, és korrekciós intézkedéseket tehetnek. A folyamatos monitoring segít biztosítani, hogy az öngyógyító adatfolyamat megbízható maradjon és összhangban legyen a kívánt eredményekkel.

A megmagyarázhatóság és az átláthatóság kritikus szempontok az öngyógyító adattechnikák megvalósításakor. Az adatjavítások világos magyarázatával, átfogó ellenőrzési nyomvonalak fenntartásával és az emberi felügyelet bevonásával a szervezetek bizalmat építhetnek az öngyógyító adatfolyamatban, és biztosíthatják, hogy az adatokon végzett módosítások indokolhatók és összhangban vannak az üzleti célkitűzésekkel.

Fontos megtalálni az egyensúlyt az automatizálás előnyei és az átláthatóság szükségessége között. Bár az öngyógyító adattechnikák jelentősen javíthatják az adatminőséget és a hatékonyságot, ez nem mehet az adatjavítási folyamat láthatóságának és ellenőrzésének rovására. Az öngyógyító adatfeldolgozók megmagyarázhatóságot és átláthatóságot szem előtt tartó tervezésével a szervezetek kiaknázhatják az AI erejét, miközben fenntartják az adatok szükséges elszámoltathatósági és bizalmi szintjét.

Nem szándékolt következmények

Bár az öngyógyító adattechnikák célja az adatminőség és konzisztencia javítása, fontos tudatában lenni a nem szándékolt következmények lehetőségének. Az

automatizált javítások, ha nem körültekintően tervezik és felügyelik őket, akaratlanul is megváltoztathatják az adatok jelentését vagy kontextusát, ami következményes problémákhoz vezethet.

Az öngyógyító adatok egyik fő kockázata a torzítások vagy hibák bevezetése az adatjavítási folyamatba. Az AI modellek, mint minden más szoftverrendszer, ki lehetnek téve a tanítóadatokban jelenlévő vagy az algoritmusok tervezése során bevezetett torzításoknak. Ha ezeket a torzításokat nem azonosítják és nem mérséklik, azok továbbterjedhetnek az öngyógyító adatfolyamaton keresztül, és torz vagy helytelen adatmódosításokat eredményezhetnek.

Vegyünk például egy öngyógyító adatfeldolgozót, amelynek feladata az ügyfelek demográfiai adataiban lévő következetlenségek javítása. Ha a mesterséges intelligencia modell elfogultságokat tanult a történeti adatokból, például bizonyos foglalkozásokat vagy jövedelmi szinteket összekapcsol meghatározott nemekkel vagy etnikai hovatartozással, akkor helytelen feltételezéseket tehet, és olyan módon módosíthatja az adatokat, amely megerősíti ezeket az elfogultságokat. Ez pontatlan ügyfélprofilokhoz, félrevezető üzleti döntésekhez és potenciálisan diszkriminatív eredményekhez vezethet.

Egy másik lehetséges nem szándékolt következmény az értékes információk vagy kontextus elvesztése az adatjavítási folyamat során. Az öngyógyító adattechnikák gyakran az adatok szabványosítására és normalizálására összpontosítanak a következetesség biztosítása érdekében. Azonban egyes esetekben az eredeti adatok olyan árnyalatokat, kivételeket vagy kontextuális információkat tartalmazhatnak, amelyek fontosak a teljes kép megértéséhez. Az automatizált javítások, amelyek vakon kényszerítik a szabványosítást, akaratlanul is eltávolíthatják vagy elhomályosíthatják ezeket az értékes információkat.

Képzeljünk el például egy öngyógyító adatfeldolgozót, amely az orvosi nyilvántartásokban lévő következetlenségek javításáért felelős. Ha a feldolgozó olyan beteg kórtörténetével találkozik, akinél ritka betegség vagy szokatlan kezelési terv szerepel, megpróbálhatja normalizálni az adatokat, hogy azok jobban illeszkedjenek

egy gyakoribb mintázathoz. Azonban ezzel elveszhetnek azok a specifikus részletek és kontextus, amelyek kulcsfontosságúak a beteg egyedi helyzetének pontos ábrázolásához. Ez az információvesztés súlyos következményekkel járhat a betegellátás és az orvosi döntéshozatal szempontjából.

A nem szándékolt következmények kockázatának csökkentése érdekében elengedhetetlen a proaktív megközelítés az öngyógyító adattechnikák tervezése és megvalósítása során:

1. **Alapos tesztelés és validálás**: Az öngyógyító adatfeldolgozók éles környezetbe való telepítése előtt kulcsfontosságú viselkedésük alapos tesztelése és validálása különböző forgatókönyvek mellett. Ez magában foglalja a tesztelést olyan reprezentatív adatkészletekkel, amelyek lefedik a különböző szélsőséges eseteket, kivételeket és potenciális elfogultságokat. Az alapos tesztelés segít azonosítani és kezelni a nem szándékolt következményeket, mielőtt azok hatással lennének a valós adatokra.
2. **Folyamatos monitoring és értékelés**: Elengedhetetlen a folyamatos monitoring és értékelési mechanizmusok bevezetése a nem szándékolt következmények időbeni észlelése és enyhítése érdekében. Az öngyógyító adatfolyamatok eredményeinek rendszeres felülvizsgálata, a kapcsolódó rendszerekre és döntéshozatalra gyakorolt hatás elemzése, valamint az érintettek visszajelzéseinek gyűjtése segíthet azonosítani a káros hatásokat és időben korrekciós intézkedéseket kezdeményezni. Ha szervezetének vannak működési irányítópultjai, valószínűleg jó ötlet egyértelműen látható mérőszámokat hozzáadni az automatizált adatváltozásokhoz kapcsolódóan. Még jobb ötlet lehet riasztásokat beállítani a normál adatváltozási aktivitástól való jelentős eltérések esetére!
3. **Emberi felügyelet és beavatkozás**: Kulcsfontosságú az emberi felügyelet és a beavatkozás lehetőségének fenntartása az öngyógyító adatfolyamatban. Bár az automatizálás nagyban javíthatja a hatékonyságot, fontos, hogy

szakértők felülvizsgálják és validálják a mesterséges intelligencia modellek által végzett javításokat, különösen kritikus vagy érzékeny területeken. Az emberi ítélőképesség és szakterületi szakértelem segíthet azonosítani és kezelni az esetlegesen felmerülő nem szándékolt következményeket.

4. **Magyarázható mesterséges intelligencia és átláthatóság**: Ahogy az előző alfejezetben tárgyaltuk, a magyarázható mesterséges intelligencia technikák beépítése és az átláthatóság biztosítása az öngyógyító adatfolyamatban segíthet enyhíteni a nem szándékolt következményeket. Az adatjavítások világos magyarázatával és átfogó ellenőrzési naplók vezetésével a szervezetek jobban megérthetik és nyomon követhetik a mesterséges intelligencia modellek által végzett módosítások mögötti indokokat.
5. **Fokozatos és iteratív megközelítés**: Az öngyógyító adatok fokozatos és iteratív megközelítésének alkalmazása segíthet minimalizálni a nem szándékolt következmények kockázatát. Ahelyett, hogy az automatizált javításokat egyszerre alkalmaznánk a teljes adatkészletre, kezdjük az adatok egy részhalmazával, és fokozatosan bővítsük a hatókört, ahogy a technikák hatékonynak és megbízhatónak bizonyulnak. Ez lehetővé teszi a gondos megfigyelést és módosítást menet közben, csökkentve a nem szándékolt következmények hatását.
6. **Együttműködés és visszajelzés**: A különböző területek érintettjeinek bevonása és az együttműködés, valamint a visszajelzések ösztönzése az öngyógyító adatfolyamat során segíthet azonosítani és kezelni a nem szándékolt következményeket. A szakterületi szakértők, adatfelhasználók és végfelhasználók rendszeres visszajelzéseinek kérése értékes betekintést nyújthat az adatjavítások valós hatásába, és rávilágíthat az esetlegesen figyelmen kívül hagyott problémákra.

A nem szándékolt következmények kockázatának proaktív kezelésével és megfelelő biztosítékok bevezetésével a szervezetek kihasználhatják az öngyógyító adattechnikák előnyeit, miközben minimalizálják a lehetséges káros hatásokat. Fontos, hogy az

öngyógyító adatokat iteratív és együttműködésen alapuló folyamatként kezeljük, folyamatosan monitorozva, értékelve és finomítva a technikákat annak biztosítása érdekében, hogy összhangban legyenek a kívánt eredményekkel és fenntartsák az adatok integritását és megbízhatóságát.

Az öngyógyító adatminták használatának mérlegelésekor elengedhetetlen ezen tényezők gondos értékelése és az előnyök összevetése a potenciális kockázatokkal és korlátokkal. Egyes esetekben egy hibrid megközelítés, amely ötvözi az automatizált javításokat az emberi felügyelettel és beavatkozással, lehet a legmegfelelőbb megoldás.

Érdemes megjegyezni, hogy az öngyógyító adattechnikák nem helyettesíthetik a robusztus adatellenőrzési, bemeneti tisztítási és hibakezelési mechanizmusokat. Ezek az alapvető gyakorlatok továbbra is kritikus fontosságúak az adatok integritásának és biztonságának biztosításához. Az öngyógyító adatokra úgy kell tekinteni, mint egy kiegészítő megközelítésre, amely bővítheti és fejlesztheti ezeket a meglévő intézkedéseket.

Végső soron az öngyógyító adatminták alkalmazására vonatkozó döntés az alkalmazás specifikus követelményeitől, korlátaitól és prioritásaitól függ. A fent vázolt megfontolások gondos mérlegelésével és az alkalmazás céljaival és architektúrájával való összehangolásával megalapozott döntéseket hozhat arról, hogy mikor és hogyan használja hatékonyan az öngyógyító adattechnikákat.

Kontextuális Tartalomgenerálás

A Kontextuális Tartalomgenerálási minták kihasználják a nagy nyelvi modellek (NNY) erejét az alkalmazásokon belüli dinamikus és kontextusfüggő tartalom létrehozásához. A minták ezen kategóriája felismeri annak fontosságát, hogy személyre szabott és releváns tartalmat nyújtson a felhasználóknak specifikus igényeik, preferenciáik, sőt korábbi és jelenlegi alkalmazáshasználatuk alapján.

> Ebben a megközelítésben a “tartalom” kifejezés mind az elsődleges tartalomra (pl. blogbejegyzések, cikkek stb.), mind a metatartalomra, például az elsődleges tartalomhoz kapcsolódó ajánlásokra vonatkozik.

A Kontextuális Tartalomgenerálási minták kulcsszerepet játszhatnak a felhasználói elkötelezettség szintjének növelésében, személyre szabott élmények nyújtásában és

a tartalomkészítési feladatok automatizálásában mind az Ön, mind a felhasználói számára. Az ebben a fejezetben leírt minták alkalmazásával olyan alkalmazásokat hozhat létre, amelyek dinamikusan generálnak tartalmat, valós időben alkalmazkodva a kontextushoz és a bemenetekhez.

A minták a NNY-ek integrálásával működnek az alkalmazás kimenetein, a felhasználói felülettől (amit néha "chrome"-nak neveznek) kezdve az e-maileken és egyéb értesítéseken át a tartalomgenerálási folyamatokig.

Amikor egy felhasználó interakcióba lép az alkalmazással vagy egy konkrét tartalomkérést indít, az alkalmazás rögzíti a releváns kontextust, például a felhasználói preferenciákat, korábbi interakciókat vagy specifikus promptokat. Ezt a kontextuális információt aztán a NNY-be táplálják, minden szükséges sablonnal vagy irányelvvel együtt, és olyan szöveges kimenet előállítására használják, amit egyébként vagy keményen kódolni kellene, vagy adatbázisban tárolni, vagy algoritmikusan generálni.

A NNY által generált tartalom különböző formákat ölthet, például személyre szabott ajánlásokat, dinamikus termékleírásokat, testreszabott e-mail válaszokat, vagy akár teljes cikkeket és blogbejegyzéseket. Ennek a tartalomnak az egyik legradikálisabb felhasználási módja, amit több mint egy éve úttörőként vezettem be, a felhasználói felület elemeinek dinamikus generálása, például űrlapcímkék, elemleírások és egyéb magyarázó szövegek formájában.

Személyre szabás

A Kontextuális Tartalomgenerálási minták egyik fő előnye a felhasználók számára nyújtott, nagymértékben személyre szabott élmények létrehozásának képessége. A felhasználó-specifikus kontextus alapján történő tartalomgenerálással ezek a minták lehetővé teszik az alkalmazások számára, hogy a tartalmat az egyes felhasználók érdeklődési köréhez, preferenciáihoz és interakcióihoz igazítsák.

A személyre szabás túlmutat azon, hogy egyszerűen beillesztjük a felhasználó nevét

egy általános tartalomba. Magában foglalja minden egyes felhasználóról rendelkezésre álló gazdag kontextus kihasználását olyan tartalom generálása érdekében, amely rezonál specifikus igényeikkel és vágyaikkal. Ez a kontextus számos tényezőt tartalmazhat, például:

1. **Felhasználói Profil Információk**: Ezen technika alkalmazásának legáltalánosabb szintjén a demográfiai adatok, érdeklődési körök, preferenciák és egyéb profilattribútumok felhasználhatók olyan tartalom generálására, amely illeszkedik a felhasználó hátteréhez és jellemzőihez.
2. **Viselkedési Adatok**: A felhasználó alkalmazással való korábbi interakciói, például a megtekintett oldalak, kattintott linkek vagy megvásárolt termékek értékes betekintést nyújthatnak viselkedésükbe és érdeklődési körükbe. Ezek az adatok felhasználhatók olyan tartalomjavaslatok generálására, amelyek tükrözik elkötelezettségi mintáikat és előrejelzik jövőbeli igényeiket.
3. **Kontextuális Tényezők**: A felhasználó aktuális kontextusa, például helyzete, eszköze, a napszak vagy akár az időjárás befolyásolhatja a tartalomgenerálási folyamatot. Például egy utazási alkalmazás rendelkezhet olyan AI munkamenettel, amely képes személyre szabott ajánlásokat generálni a felhasználó aktuális helyzete és az uralkodó időjárási körülmények alapján.

Ezen kontextuális tényezők kihasználásával a Kontextuális Tartalomgenerálási minták lehetővé teszik az alkalmazások számára olyan tartalom szolgáltatását, amely minden egyes felhasználó számára személyre szabottnak tűnik. A személyre szabás ezen szintje több jelentős előnnyel jár:

1. **Növelt Elkötelezettség**: A személyre szabott tartalom megragadja a felhasználók figyelmét és fenntartja elkötelezettségüket az alkalmazás iránt. Amikor a felhasználók úgy érzik, hogy a tartalom releváns és közvetlenül az ő igényeikhez szól, nagyobb valószínűséggel töltenek több időt az alkalmazással való interakcióval és funkcióinak felfedezésével.

2. **Javított Felhasználói Elégedettség**: A személyre szabott tartalom demonstrálja, hogy az alkalmazás megérti és törődik a felhasználó egyedi igényeivel. Olyan tartalom szolgáltatásával, amely hasznos, informatív és összhangban van érdeklődési körükkel, az alkalmazás növelheti a felhasználói elégedettséget és erősebb kapcsolatot építhet ki felhasználóival.
3. **Magasabb Konverziós Ráták**: E-kereskedelmi vagy marketing alkalmazások esetében a személyre szabott tartalom jelentősen befolyásolhatja a konverziós rátákat. Azáltal, hogy a felhasználóknak olyan termékeket, ajánlatokat vagy ajánlásokat mutat be, amelyek preferenciáikra és viselkedésükre vannak szabva, az alkalmazás növelheti annak valószínűségét, hogy a felhasználók végrehajtják a kívánt műveleteket, például vásárlást vagy szolgáltatásra való feliratkozást.

Produktivitás

A Kontextuális Tartalomgenerálási minták jelentősen növelhetik bizonyos típusú produktivitást azáltal, hogy csökkentik a manuális tartalomgenerálás és szerkesztés szükségességét a kreatív folyamatokban. A NNY-ek erejének kihasználásával nagy mennyiségben generálhat minőségi tartalmat, időt és energiát megtakarítva, amit a tartalomkészítőknek és fejlesztőknek egyébként unalmas kézi munkára kellene fordítaniuk.

Hagyományosan a tartalomkészítőknek kutatniuk, írniuk, szerkeszteniük és formázniuk kell a tartalmakat, hogy azok megfeleljenek az alkalmazás követelményeinek és a felhasználói elvárásoknak. Ez a folyamat időigényes és erőforrás-intensive lehet, különösen ahogy a tartalom mennyisége növekszik.

Azonban a Kontextuális Tartalomgenerálási mintákkal a tartalomkészítési folyamat nagyban automatizálható. Az LLM-ek képesek összefüggő, nyelvtanilag helyes és kontextuálisan releváns tartalmakat generálni a megadott promptok és irányelvek alapján. Ez az automatizálás több termelékenységi előnnyel jár:

1. **Csökkentett Manuális Munka**: Azáltal, hogy a tartalomgenerálási feladatokat LLM-ekre bízzuk, a tartalomkészítők magasabb szintű feladatokra összpontosíthatnak, mint például tartalomstratégia, ötletelés és minőségbiztosítás. Megadhatják a szükséges kontextust, sablonokat és irányelveket az LLM-nek, és hagyhatják, hogy az kezelje a tényleges tartalomgenerálást. Ez csökkenti az íráshoz és szerkesztéshez szükséges manuális munkát, lehetővé téve, hogy a tartalomkészítők produktívabbak és hatékonyabbak legyenek.
2. **Gyorsabb Tartalomkészítés**: Az LLM-ek sokkal gyorsabban tudnak tartalmat generálni, mint az emberi írók. A megfelelő promptokkal és irányelvekkel egy LLM másodpercek vagy percek alatt több tartalmat is képes létrehozni. Ez a sebesség lehetővé teszi, hogy az alkalmazások sokkal gyorsabban generáljanak tartalmat, lépést tartva a felhasználók igényeivel és az állandóan változó digitális környezettel.

> Vajon a gyorsabb tartalomkészítés a "közlegelők tragédiája" helyzethez vezet, ahol az internet megfullad a senki által nem olvasott tartalomtól? Sajnos, azt gyanítom, hogy igen.

3. **Konzisztencia és Minőség**: Az LLM-ek könnyedén tudják úgy átdolgozni a tartalmat, hogy az stílusában, hangnemében és minőségében következetes legyen. Világos irányelvek és példák mellett bizonyos típusú alkalmazások (pl. hírszerkesztőségek, PR stb.) biztosíthatják, hogy az ember által generált tartalmuk összhangban legyen a márkahangjukkal és megfeleljen a kívánt minőségi standardoknak. Ez a következetesség csökkenti a kiterjedt szerkesztés és átdolgozás szükségességét, időt és erőfeszítést megtakarítva a tartalomkészítési folyamatban.
4. **Iteráció és Optimalizálás**: A Kontextuális Tartalomgenerálási minták lehetővé teszik a tartalom gyors iterációját és optimalizálását. A promptok, sablonok

vagy irányelvek módosításával az alkalmazások gyorsan generálhatnak tartalomvariációkat és tesztelhetnek különböző megközelítéseket olyan automatizált módon, ami korábban soha nem volt lehetséges. Ez az iteratív folyamat gyorsabb kísérletezést és a tartalomstratégiák finomítását teszi lehetővé, ami idővel hatékonyabb és vonzóbb tartalomhoz vezet. Ez a technika különösen forradalmi lehet olyan alkalmazások esetében, mint az e-kereskedelem, amelyek sikere a visszafordulási arányokon és a felhasználói interakción múlik

Fontos megjegyezni, hogy bár a Kontextuális Tartalomgenerálási minták nagyban növelhetik a produktivitást, nem szüntetik meg teljesen az emberi közreműködés szükségességét. A tartalomkészítők és szerkesztők továbbra is kulcsfontosságú szerepet játszanak az átfogó tartalomstratégia meghatározásában, az LLM irányításában és a generált tartalom minőségének és megfelelőségének biztosításában.

A tartalomkészítés ismétlődőbb és időigényesebb aspektusainak automatizálásával a Kontextuális Tartalomgenerálási minták felszabadítják az értékes emberi időt és erőforrásokat, amelyek így magasabb értékű feladatokra irányíthatók. Ez a megnövekedett produktivitás lehetővé teszi, hogy személyre szabottabb és vonzóbb tartalmat nyújtsunk a felhasználóknak, miközben optimalizáljuk a tartalomkészítési munkafolyamatokat.

Gyors Iteráció és Kísérletezés

A Kontextuális Tartalomgenerálási minták lehetővé teszik a különböző tartalomvariációkkal való gyors iterációt és kísérletezést, ami gyorsabb optimalizálást és a tartalomstratégia finomítását teszi lehetővé. Másodpercek alatt több tartalomverziót generálhatsz, egyszerűen a modellnek megadott kontextus, sablonok vagy irányelvek módosításával.

Ez a gyors iterációs képesség több kulcsfontosságú előnnyel jár:

1. **Tesztelés és Optimalizálás**: A gyors tartalomvariáció-generálási képességgel könnyen tesztelhetsz különböző megközelítéseket és mérheted azok hatékonyságát. Például generálhatsz több verziót egy termékleírásból vagy marketing üzenetből, mindegyiket egy adott felhasználói szegmensre vagy kontextusra szabva. A felhasználói interakciós metrikák, mint például az átkattintási arányok vagy konverziós ráták elemzésével azonosíthatod a leghatékonyabb tartalomvariációkat és ennek megfelelően optimalizálhatod a tartalomstratégiádat.

2. **A/B Tesztelés**: A Kontextuális Tartalomgenerálási minták zökkenőmentes A/B tesztelést tesznek lehetővé. Generálhatsz két vagy több tartalomvariációt és véletlenszerűen szolgálhatod ki őket különböző felhasználói csoportoknak. Az egyes variációk teljesítményének összehasonlításával meghatározhatod, hogy melyik tartalom rezonál legjobban a célközönségeddel. Ez az adatvezérelt megközelítés lehetővé teszi, hogy megalapozott döntéseket hozz és folyamatosan finomítsd a tartalmadat a felhasználói interakció maximalizálása és a kívánt eredmények elérése érdekében.
3. **Személyre Szabási Kísérletek**: A gyors iteráció és kísérletezés különösen értékes a személyre szabás terén. A Kontextuális Tartalomgenerálási mintákkal gyorsan generálhatsz személyre szabott tartalomvariációkat különböző felhasználói szegmensek, preferenciák vagy viselkedések alapján. A különböző személyre szabási stratégiákkal való kísérletezés révén azonosíthatod a leghatékonyabb megközelítéseket az egyéni felhasználók megszólítására és a testreszabott élmények nyújtására.
4. **Alkalmazkodás a változó trendekhez**: A gyors iteráció és kísérletezés képessége lehetővé teszi, hogy agilis maradj és alkalmazkodj a változó trendekhez és felhasználói preferenciákhoz. Ahogy új témák, kulcsszavak vagy felhasználói viselkedésminták jelennek meg, gyorsan létrehozhatsz olyan tartalmakat, amelyek illeszkednek ezekhez a trendekhez. A tartalmak folyamatos kísérletezésével és

finomításával releváns maradhatsz és megőrizheted versenyelőnyödet a folyamatosan fejlődő digitális környezetben.

5. **Költséghatékony kísérletezés**: A hagyományos tartalmi kísérletezés gyakran jelentős időt és erőforrást igényel, mivel a tartalomkészítőknek manuálisan kell kifejleszteniük és tesztelniük a különböző változatokat. Azonban a Kontextuális Tartalomgenerálási mintákkal a kísérletezés költsége jelentősen csökken. A nagy nyelvi modellek gyorsan és nagy méretekben képesek tartalomváltozatokat generálni, lehetővé téve számos ötlet és megközelítés feltárását jelentős költségek nélkül.

A gyors iteráció és kísérletezés maximális kihasználásához fontos, hogy rendelkezz egy jól meghatározott kísérletezési keretrendszerrel. Ennek a keretrendszernek tartalmaznia kell:

- Világos célkitűzéseket és hipotéziseket minden kísérlethez
- Megfelelő mérőszámokat és nyomonkövetési mechanizmusokat a tartalom teljesítményének méréséhez
- Szegmentációs és célzási stratégiákat annak biztosítására, hogy a releváns tartalomváltozatok a megfelelő felhasználókhoz jussanak el
- Elemzési és jelentéskészítési eszközöket a kísérleti adatokból származó betekintések kinyeréséhez
- Egy folyamatot a tanulságok és optimalizációk tartalomstratégiába való beépítéséhez

A gyors iteráció és kísérletezés elfogadásával folyamatosan finomíthatod és optimalizálhatod a tartalmadat, biztosítva, hogy az továbbra is lebilincselő, releváns és hatékony maradjon az alkalmazásod céljainak elérésében. Ez az agilis megközelítés a tartalomkészítésben lehetővé teszi, hogy az élen maradj és kivételes felhasználói élményt nyújts.

Skálázhatóság és hatékonyság

Ahogy az alkalmazások növekednek és a személyre szabott tartalom iránti igény nő, a kontextuális tartalomgenerálási minták lehetővé teszik a tartalomkészítés hatékony skálázását. A nagy nyelvi modellek képesek egyszerre több felhasználó és kontextus számára tartalmat generálni, anélkül, hogy arányosan növelni kellene az emberi erőforrásokat. Ez a skálázhatóság lehetővé teszi az alkalmazások számára, hogy személyre szabott élményt nyújtsanak egy növekvő felhasználói bázisnak anélkül, hogy megterhelnék tartalomkészítési képességeiket.

Megjegyzendő, hogy a kontextuális tartalomgenerálás hatékonyan használható az alkalmazásod "menet közbeni" nemzetköziesítéséhez. Valójában pontosan ezt tettem az Instant18n Gem segítségével, hogy az Olympiát több mint féltucatnyi nyelven elérhetővé tegyem, annak ellenére, hogy még egy évnél fiatalabbak vagyunk.

AI-alapú honosítás

Ha megengeded, hogy egy kicsit dicsekedjek, úgy gondolom, hogy a Rails alkalmazásokhoz készített Instant18n könyvtáram egy úttörő példája a "Kontextuális Tartalomgenerálás" mintának a gyakorlatban, amely bemutatja az AI transzformatív potenciálját az alkalmazásfejlesztésben. Ez a gem az OpenAI GPT nagy nyelvi modelljének erejét használja fel, hogy forradalmasítsa a nemzetköziesítés és honosítás kezelését a Rails alkalmazásokban.

Hagyományosan egy Rails alkalmazás nemzetköziesítése magában foglalja a fordítási kulcsok manuális definiálását és a megfelelő fordítások biztosítását minden támogatott nyelvhez. Ez a folyamat időigényes, erőforrás-intenzív és következetlenségekre hajlamos lehet. Az Instant18n gem azonban teljesen újradefiniálja a honosítás paradigmáját.

A nagy nyelvi modell integrálásával az Instant18n gem lehetővé teszi a fordítások menet közbeni generálását, a szöveg kontextusa és jelentése alapján. Ahelyett, hogy előre meghatározott fordítási kulcsokra és statikus fordításokra támaszkodna, a gem dinamikusan fordítja a szöveget az AI erejét használva. Ez a megközelítés több kulcsfontosságú előnyt kínál:

1. **Zökkenőmentes honosítás**: Az Instant18n gemmel a fejlesztőknek többé nem kell manuálisan definiálniuk és karbantartaniuk a fordítási fájlokat minden támogatott nyelvhez. A gem automatikusan generálja a fordításokat a megadott szöveg és a kívánt célnyelv alapján, így a honosítási folyamat erőfeszítés nélkülivé és zökkenőmentessé válik.
2. **Kontextuális pontosság**: Az AI-nak elegendő kontextus adható a fordítandó szöveg árnyalatainak megértéséhez. Figyelembe tudja venni a környező kontextust, idiómákat és kulturális utalásokat, hogy pontos, természetesen hangzó és kontextuálisan megfelelő fordításokat generáljon.
3. **Kiterjedt nyelvtámogatás**: Az Instant18n gem a GPT széles körű tudását és nyelvi képességeit használja ki, lehetővé téve a fordítást számos nyelvre. A gyakori nyelvektől, mint a spanyol és francia, egészen az obscurusabb vagy fiktív nyelvekig, mint a klingon és a tünde, a gem képes kezelni a legkülönfélébb fordítási követelményeket.
4. **Rugalmasság és kreativitás**: A gem túlmutat a hagyományos nyelvi fordításokon, és lehetővé teszi a kreatív és nem konvencionális honosítási opciókat. A fejlesztők különböző stílusokba, dialektusokba vagy akár fiktív nyelvekre is fordíthatnak szöveget, új lehetőségeket nyitva az egyedi felhasználói élmények és lebilincselő tartalmak előtt.
5. **Teljesítmény optimalizálás**: Az Instant18n gem gyorsítótárazási mechanizmusokat tartalmaz a teljesítmény javítása és az ismétlődő fordítások többletterhelésének csökkentése érdekében. A lefordított szöveg gyorsítótárba kerül, lehetővé téve, hogy az ugyanazon fordítás későbbi kérései gyorsan kiszolgálhatók legyenek redundáns API-hívások nélkül.

Az Instant18n gem példázza a “Kontextuális Tartalomgenerálás” minta erejét azáltal, hogy az AI-t használja a honosított tartalom dinamikus generálásához. Bemutatja, hogyan integrálható az AI egy Rails alkalmazás alapvető funkcionalitásába, átalakítva azt, ahogyan a fejlesztők megközelítik a nemzetköziesítést és honosítást.

A kézi fordításkezelés szükségességének kiküszöbölésével és a kontextus-alapú menet közbeni fordítások lehetővé tételével az Instant18n gem jelentős időt és erőfeszítést takarít meg a fejlesztők számára. Lehetővé teszi számukra, hogy az alkalmazásuk alapvető funkcióinak fejlesztésére összpontosítsanak, miközben biztosítja, hogy a lokalizációs aspektus zökkenőmentesen és pontosan legyen kezelve.

A felhasználói tesztelés és visszajelzés fontossága

Végül, mindig tartsa szem előtt a felhasználói tesztelés és visszajelzés fontosságát. Kulcsfontosságú annak validálása, hogy a kontextuális tartalomgenerálás megfelel-e a felhasználói elvárásoknak és összhangban van-e az alkalmazás céljaival. Folyamatosan iteráljon és finomítsa a generált tartalmat a felhasználói visszajelzések és az analitika alapján. Ha olyan nagy léptékű dinamikus tartalmat generál, amelyet Ön és csapata képtelen lenne manuálisan validálni, fontolja meg olyan visszajelzési mechanizmusok beépítését, amelyek lehetővé teszik a felhasználók számára, hogy jelenthessék a furcsa vagy hibás tartalmakat, magyarázattal együtt. Ezek az értékes visszajelzések akár egy olyan MI munkamenetbe is továbbíthatók, amelynek feladata a tartalmat generáló komponens módosítása!

Generatív UI

A figyelem napjainkban olyan értékes, hogy a hatékony felhasználói elköteleződés már nemcsak zökkenőmentes és intuitív, hanem az egyéni igényekhez, preferenciákhoz és kontextusokhoz nagymértékben személyre szabott szoftverélményeket követel meg. Ennek eredményeként a tervezők és fejlesztők egyre inkább szembesülnek azzal a kihívással, hogy olyan felhasználói felületeket kell létrehozniuk, amelyek *nagy léptékben* képesek alkalmazkodni és kiszolgálni minden felhasználó egyedi igényeit.

A Generatív UI (GenUI) egy valóban forradalmi megközelítés a felhasználói felület tervezésében, amely a nagy nyelvi modellek (LLM-ek) erejét használja fel, hogy azonnal létrehozzon erősen személyre szabott és dinamikus felhasználói élményeket. Mindenképpen szerettem volna legalább egy alapszintű bevezetést adni a GenUI-ról ebben a könyvben, mivel úgy vélem, hogy ez az egyik legzöldebb zöldmezős lehetőség, amely jelenleg létezik az alkalmazástervezés és keretrendszerek területén. Meggyőződésem, hogy tucatnyi vagy még több sikeres kereskedelmi és nyílt forráskódú

projekt fog megjelenni ebben a speciális piaci szegmensben.

Lényegét tekintve a GenUI ötvözi a Kontextuális Tartalomgenerálás elveit a fejlett AI technikákkal, hogy dinamikusan generáljon felhasználói felületi elemeket, például szöveget, képeket és elrendezéseket, a felhasználó kontextusának, preferenciáinak és céljainak mélyreható megértése alapján. A GenUI lehetővé teszi a tervezők és fejlesztők számára, hogy olyan felületeket hozzanak létre, amelyek alkalmazkodnak és fejlődnek a felhasználói interakciók alapján, olyan szintű személyre szabást biztosítva, amely korábban elérhetetlen volt.

A GenUI alapvető változást jelent a felhasználói felület tervezésének megközelítésében. Ahelyett, hogy a tömegeknek terveznénk, a GenUI lehetővé teszi, hogy az egyénnek tervezzünk. A személyre szabott tartalom és felületek olyan felhasználói élményeket képesek létrehozni, amelyek mélyebb szinten rezonálnak minden egyes felhasználóval, növelve az elköteleződést, elégedettséget és lojalitást.

Mint élvonalbeli technika, a GenUI-ra való átállás tele van koncepcionális és gyakorlati kihívásokkal. Az AI integrálása a tervezési folyamatba, annak biztosítása, hogy a generált felületek ne csak személyre szabottak, hanem használhatóak, hozzáférhetőek és összhangban legyenek az általános márkával és felhasználói élménnyel - mindezek olyan kihívások, amelyek a GenUI-t inkább kevesek, mint sokak számára teszik elérhetővé. Ráadásul az AI bevonása adatvédelmi, átláthatósági és talán még etikai kérdéseket is felvet.

A kihívások ellenére a nagy léptékű személyre szabott élmények képesek teljesen átalakítani azt, ahogyan a digitális termékekkel és szolgáltatásokkal kapcsolatba lépünk. Lehetőségeket nyit befogadó és hozzáférhető felületek létrehozására, amelyek kiszolgálják a felhasználók különböző igényeit, képességeiktől, hátterüktől vagy preferenciáiktól függetlenül.

Ebben a fejezetben felfedezzük a GenUI koncepcióját, megvizsgálva néhány meghatározó jellemzőjét, főbb előnyeit és lehetséges kihívásait. Azzal kezdjük, hogy megvizsgáljuk a GenUI legalapvetőbb és legkönnyebben hozzáférhető formáját: a

szövegek generálását egyébként hagyományosan tervezett és implementált felhasználói felületekhez.

Szövegek generálása felhasználói felületekhez

Az alkalmazás felhasználói felületének elemein megjelenő szövegek, mint például az űrlapmezők címkéi, elemleírások és magyarázó szövegek, általában közvetlenül a sablonokba vagy UI komponensekbe vannak kódolva, így minden felhasználó számára egységes, de általános élményt nyújtanak. A kontextuális tartalomgenerálási mintákat használva ezeket a statikus elemeket dinamikus, kontextus-tudatos és személyre szabott komponensekké alakíthatja.

Személyre szabott űrlapok

Az űrlapok a web- és mobilalkalmazások mindenütt jelenlévő részei, amelyek a felhasználói input gyűjtésének elsődleges eszközeiként szolgálnak. Azonban a hagyományos űrlapok gyakran általános és személytelen élményt nyújtanak, szabványos címkékkel és mezőkkel, amelyek nem mindig illeszkednek a felhasználó specifikus kontextusához vagy igényeihez. A felhasználók nagyobb valószínűséggel töltenek ki olyan űrlapokat, amelyek az igényeikhez és preferenciáikhoz igazodnak, ami magasabb konverziós rátához és felhasználói elégedettséghez vezet.

Fontos azonban egyensúlyt teremteni a személyre szabás és a következetesség között. Bár az űrlapok egyéni felhasználókhoz való igazítása előnyös lehet, alapvető fontosságú bizonyos fokú ismerősség és kiszámíthatóság megőrzése. A felhasználóknak továbbra is könnyen fel kell ismerniük és navigálniuk az űrlapokat, még a személyre szabott elemek mellett is.

Íme néhány személyre szabott űrlap ötlet inspirációként:

Kontextuális mezőjavaslatok

A GenUI elemzi a felhasználó korábbi interakcióit, preferenciáit és adatait, hogy intelligens mezőjavaslatokat nyújtson előrejelzésként. Például, ha a felhasználó korábban már megadta a szállítási címét, az űrlap automatikusan kitöltheti a vonatkozó mezőket a mentett információkkal. Ez nemcsak időt takarít meg, hanem azt is demonstrálja, hogy az alkalmazás érti és emlékszik a felhasználó preferenciáira.

Várjunk csak, ezt a technikát nem lehetne MI nélkül is megvalósítani? Természetesen igen, de az MI-vel történő megvalósítás szépsége két dologban rejlik: 1) milyen egyszerűen implementálható és 2) mennyire ellenálló marad, miközben a felhasználói felület változik és fejlődik az idő során.

Rögtönözzünk egy szolgáltatást az elméleti rendeléskezelő rendszerünkhöz, amely proaktívan próbálja kitölteni a felhasználó megfelelő szállítási címét.

```
class OrderShippingAddressSubscriber
  include Raix::ChatCompletion

  attr_accessor :order

  delegate :customer, to: :order

  DIRECTIVE = "You are a smart order processing assistant. Given the
  customer's order history, guess the most likely shipping address
  for the current order."

  def order_created(order)
    return unless order.pending? && order.shipping_address.blank?

    self.order = order

    transcript.clear
    transcript << { system: DIRECTIVE }
    transcript << { user: "Order History: #{order_history.to_json}" }
    transcript << { user: "Current Order: #{order.to_json}" }

    response = chat_completion
```

```
    apply_predicted_shipping_address(order, response)
  end

  private

  def apply_predicted_shipping_address(order, response)
    # extract the shipping address from the response...
    # ...and assume there's some sort of live update of the address fields
    order.update(shipping_address:)
  end

  def order_history
    customer.orders.successful.limit(100).map do |order|
      {
        date: order.date,
        description: order.description,
        shipping_address: order.shipping_address
      }
    end
  end
end
```

Ez egy nagyon leegyszerűsített példa, de a legtöbb esetben működnie kell. Az alapelv az, hogy hagyjuk a mesterséges intelligenciát úgy következtetni, ahogyan egy ember tenné. Hogy világosabbá tegyem, miről beszélek, nézzünk meg néhány mintaadatot:

```
Order History:
[
  {"date": "2024-01-03", "description": "garden soil mix",
   "shipping_address": "123 Country Lane, Rural Town"},
  {"date": "2024-01-15", "description": "hardcover fiction novels",
   "shipping_address": "456 City Apt, Metroville"},
  {"date": "2024-01-22", "description": "baby diapers", "shipping_address":
   "789 Suburb St, Quietville"},
  {"date": "2024-02-01", "description": "organic vegetables",
   "shipping_address": "123 Country Lane, Rural Town"},
  {"date": "2024-02-17", "description": "mystery thriller book set",
   "shipping_address": "456 City Apt, Metroville"},
  {"date": "2024-02-25", "description": "baby wipes",
   "shipping_address": "789 Suburb St, Quietville"},
```

```
  {"date": "2024-03-05", "description": "flower seeds",
   "shipping_address": "123 Country Lane, Rural Town"},
  {"date": "2024-03-20", "description": "biographies",
   "shipping_address": "456 City Apt, Metroville"},
  {"date": "2024-03-30", "description": "baby formula",
   "shipping_address": "789 Suburb St, Quietville"},
  {"date": "2024-04-12", "description": "lawn fertilizer",
   "shipping_address": "123 Country Lane, Rural Town"},
  {"date": "2024-04-22", "description": "science fiction novels",
   "shipping_address": "456 City Apt, Metroville"},
  {"date": "2024-05-02", "description": "infant toys",
   "shipping_address": "789 Suburb St, Quietville"},
  {"date": "2024-05-14", "description": "outdoor grill",
   "shipping_address": "123 Country Lane, Rural Town"},
  {"date": "2024-05-29", "description": "literary classics",
  "shipping_address": "456 City Apt, Metroville"},
  {"date": "2024-06-11", "description": "baby clothes",
   "shipping_address": "789 Suburb St, Quietville"},
  {"date": "2024-07-01", "description": "watering can",
   "shipping_address": "123 Country Lane, Rural Town"},
  {"date": "2024-07-18", "description": "non-fiction essays",
  "shipping_address": "456 City Apt, Metroville"},
  {"date": "2024-07-28", "description": "baby bath items",
  "shipping_address": "789 Suburb St, Quietville"},
  {"date": "2024-08-09", "description": "herb garden kit",
  "shipping_address": "123 Country Lane, Rural Town"},
  {"date": "2024-08-24", "description": "children's books",
  "shipping_address": "456 City Apt, Metroville"}
]
```

Észrevetted a mintát az adatokban? Garantálom, hogy ez gyerekjáték egy NNY számára. Hogy ezt demonstráljam, kérdezzük meg a GPT-4-et, mi a legvalószínűbb szállítási cím egy “hőmérő” esetében.

```
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.
```

Ha arra gondolsz, hogy túlzás egy olyan drága modellt használni erre a feladatra, mint a GPT-4, igazad van! Ugyanezt a promptot kipróbáltam a Mistral 7B Instruct modellen, és az másodpercenként 75 tokent produkált, mindössze 0,000218 USD nevetségesen alacsony költséggel.

```
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.
```

Megéri-e ennek a technikának a többletterhelése és költsége, hogy varázslatos élménnyé tegyük a fizetési folyamatot? Sok online kereskedő számára feltétlenül. És a jelek szerint az AI számítási költségei csak csökkenni fognak, különösen a nyílt forráskódú modellszolgáltatók között zajló árversenyben.

Használj Prompt Sablont és StructuredIO-t Válasz Elhatárolással együtt az ilyen chat kiegészítések optimalizálásához.

Adaptív Mezősorrend

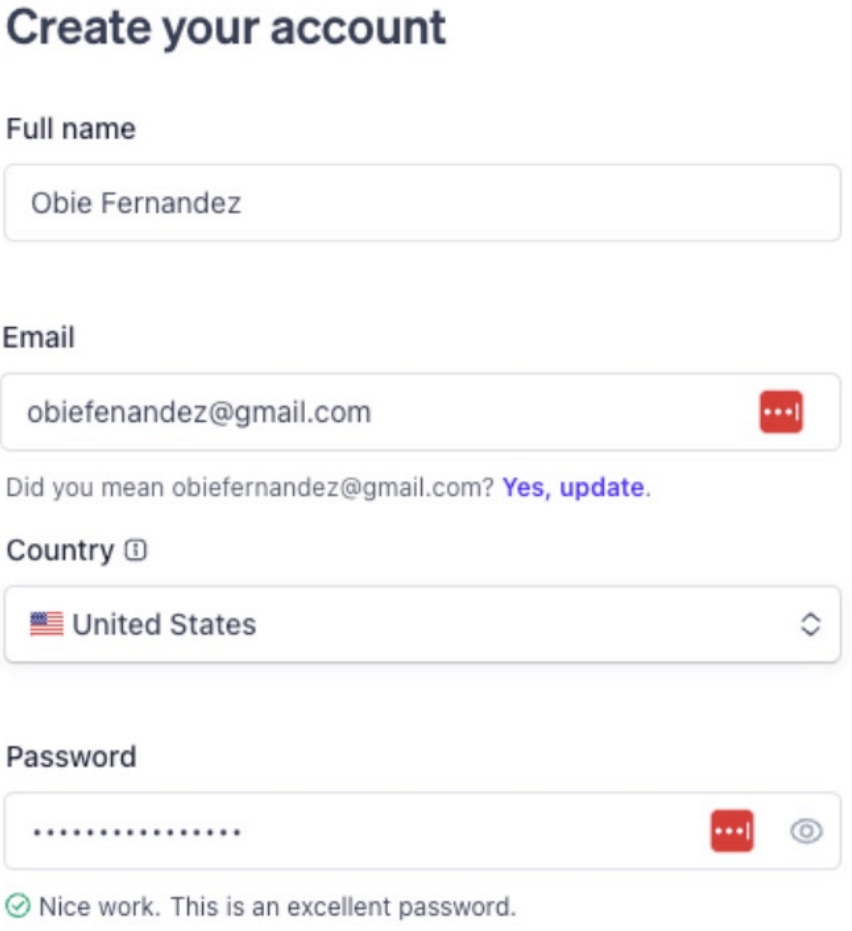
Az űrlapmezők megjelenítési sorrendje jelentősen befolyásolhatja a felhasználói élményt és a kitöltési arányokat. A GenUI segítségével dinamikusan módosíthatod a mezők sorrendjét a felhasználó kontextusa és az egyes mezők fontossága alapján. Például, ha a felhasználó egy fitneszalkalmazás regisztrációs űrlapját tölti ki, az űrlap előre helyezheti a fitneszcélokkal és preferenciákkal kapcsolatos mezőket, így téve relevánsabbá és vonzóbbá a folyamatot.

Személyre Szabott Mikrotartalom

Az űrlapokhoz tartozó útmutató szövegek, hibaüzenetek és egyéb mikrotartalmak szintén személyre szabhatók a GenUI segítségével. Az olyan általános hibaüzenetek helyett, mint az "Érvénytelen e-mail cím", generálhatsz hasznosabb és kontextuális üzeneteket, például: "Kérjük, adjon meg egy érvényes e-mail címet a rendelési visszaigazolás fogadásához." Ezek a személyre szabott elemek felhasználóbarátabbá és kevésbé frusztrálóvá tehetik az űrlapkitöltési élményt.

Személyre Szabott Validáció

A Személyre Szabott Mikrotartalomhoz hasonlóan az AI-t használhatod az űrlap varázslatos módon történő validálására. Képzeld el, hogy egy AI ellenőrzi a felhasználói profil űrlapot, *szemantikai* szinten keresve a lehetséges hibákat.



ábra 9. Észrevetted a szemantikai validációt?

Fokozatos Feltárás

A GenUI intelligensen meg tudja határozni, mely űrlapmezők lényegesek a felhasználó kontextusa alapján, és fokozatosan tárhatja fel a további mezőket szükség szerint. Ez a fokozatos feltárási technika segít csökkenteni a kognitív terhelést és kezelhetőbbé teszi az űrlapkitöltési folyamatot. Például ha egy felhasználó alapszintű előfizetésre

regisztrál, az űrlap kezdetben csak az alapvető mezőket jelenítheti meg, és ahogy a felhasználó halad előre vagy bizonyos opciókat választ, dinamikusan jelenhetnek meg további releváns mezők.

Kontextusfüggő Magyarázó Szöveg

Az elemleírásokat gyakran használjuk arra, hogy további információt vagy útmutatást nyújtsunk a felhasználóknak, amikor bizonyos elemek fölé viszik az egeret vagy interakcióba lépnek velük. A "Kontextuális Tartalomgenerálás" megközelítéssel olyan elemleírásokat generálhatsz, amelyek alkalmazkodnak a felhasználó kontextusához és releváns információkat nyújtanak. Például ha egy felhasználó egy összetett funkciót fedez fel, az elemleírás személyre szabott tippeket vagy példákat kínálhat a korábbi interakciók vagy készségszint alapján.

A magyarázó szövegek, mint például az utasítások, leírások vagy súgóüzenetek, dinamikusan generálhatók a felhasználó kontextusa alapján. Az általános magyarázatok helyett használhatsz LLM-eket olyan szövegek generálására, amelyek a felhasználó konkrét igényeihez vagy kérdéseihez igazodnak. Például ha egy felhasználó egy adott lépésnél elakad, a magyarázó szöveg személyre szabott útmutatást vagy hibaelhárítási tippeket nyújthat.

A mikrotartalom azokra a kis szövegelemekre utal, amelyek végigvezetik a felhasználókat az alkalmazáson, mint például a gombok feliratai, hibaüzenetek vagy megerősítő üzenetek. A Kontextuális Tartalomgenerálás megközelítés mikrotartalomra való alkalmazásával olyan adaptív felhasználói felületet hozhatsz létre, amely reagál a felhasználó műveleteire és releváns, hasznos szöveget biztosít. Például ha egy felhasználó kritikus műveletet készül végrehajtani, a megerősítő üzenet dinamikusan generálható, hogy világos és személyre szabott üzenetet nyújtson.

A személyre szabott magyarázó szövegek és elemleírások nagyban javíthatják az új felhasználók betanulási folyamatát. A kontextusfüggő útmutatás és példák

biztosításával segíthetsz a felhasználóknak gyorsan megérteni és navigálni az alkalmazást, csökkentve a tanulási görbét és növelve az adoptációt.

A dinamikus és kontextusfüggő felületi elemek szintén intuitívabbá és vonzóbbá tehetik az alkalmazást. A felhasználók nagyobb valószínűséggel lépnek interakcióba és fedeznek fel funkciókat, amikor a kísérő szöveg az ő konkrét igényeikhez és érdeklődésükhöz igazodik.

Eddig áttekintettük a meglévő UI paradigmák MI-vel történő fejlesztésének lehetőségeit, de mi a helyzet a felhasználói felületek tervezésének és megvalósításának radikálisabb újragondolásával?

A generatív UI meghatározása

A hagyományos UI-tervezéssel ellentétben, ahol a tervezők fix, statikus felületeket hoznak létre, a GenUI egy olyan jövőt vetít előre, amelyben szoftvereink rugalmas, személyre szabott élményeket nyújtanak, amelyek valós időben képesek fejlődni és alkalmazkodni. Valahányszor MI-vezérelt beszélgető felületet használunk, hagyjuk, hogy az MI alkalmazkodjon a felhasználó egyedi igényeihez. A GenUI egy lépéssel tovább megy azzal, hogy ezt az alkalmazkodóképességet a szoftver *vizuális* felületére is alkalmazza.

Az, hogy ma már lehet kísérletezni a GenUI ötletekkel, annak köszönhető, hogy a nagy nyelvi modellek már értik a programozást, és az alapismereteik magukban foglalják a UI technológiákat és keretrendszereket. A kérdés tehát az, hogy használhatók-e a nagy nyelvi modellek olyan UI elemek generálására, mint szövegek, képek, elrendezések, sőt teljes felületek, amelyek minden egyes felhasználóra szabottak. A modell utasítható lenne arra, hogy különböző tényezőket vegyen figyelembe, például a felhasználó korábbi interakcióit, megadott preferenciáit, demográfiai adatait és a használat aktuális kontextusát, hogy rendkívül személyre szabott és releváns felületeket hozzon létre.

A GenUI több kulcsfontosságú szempontból különbözik a hagyományos felhasználói felület tervezésétől:

1. **Dinamikus és adaptív**: A hagyományos UI-tervezés fix, statikus felületeket hoz létre, amelyek minden felhasználó számára ugyanolyanok maradnak. Ezzel szemben a GenUI olyan felületeket tesz lehetővé, amelyek dinamikusan alkalmazkodhatnak és változhatnak a felhasználói igények és kontextus alapján. Ez azt jelenti, hogy ugyanaz az alkalmazás különböző felhasználóknak különböző felületeket mutathat, sőt ugyanannak a felhasználónak is eltérő helyzetekben.
2. **Nagymértékű személyre szabás**: A hagyományos tervezésnél a személyre szabott élmények létrehozása gyakran kivitelezhetetlen az időigény és az erőforrásigény miatt. A GenUI ezzel szemben lehetővé teszi a nagymértékű személyre szabást. Az MI segítségével a tervezők olyan felületeket hozhatnak létre, amelyek automatikusan alkalmazkodnak minden felhasználó egyedi igényeihez és preferenciáihoz, anélkül hogy minden felhasználói szegmensnek külön felületet kellene tervezni és fejleszteni.
3. **Fókusz az eredményeken**: A hagyományos UI-tervezés gyakran a vizuálisan vonzó és funkcionális felületek létrehozására összpontosít. Bár ezek a szempontok a GenUI esetében is fontosak maradnak, a fő hangsúly a kívánt felhasználói eredmények elérésére helyeződik át. A GenUI célja olyan felületek létrehozása, amelyek minden felhasználó specifikus céljaira és feladataira vannak optimalizálva, a használhatóságot és a hatékonyságot előtérbe helyezve a tisztán esztétikai megfontolásokkal szemben.
4. **Folyamatos tanulás és fejlődés**: A GenUI rendszerek folyamatosan tanulhatnak és fejlődhetnek a felhasználói interakciók és visszajelzések alapján. Ahogy a felhasználók használják a generált felületeket, az MI modellek adatokat gyűjthetnek a felhasználói viselkedésről, preferenciákról és eredményekről, és ezeket az információkat felhasználhatják a jövőbeli felületgenerációk finomítására és optimalizálására. Ez az iteratív tanulási folyamat lehetővé teszi, hogy a GenUI rendszerek idővel egyre hatékonyabban elégítsék ki a felhasználói igényeket.

Fontos megjegyezni, hogy a GenUI nem azonos az MI-támogatott tervezőeszközökkel, amelyek javaslatokat adnak vagy bizonyos tervezési feladatokat automatizálnak. Bár ezek az eszközök hasznosak lehetnek a tervezési folyamat egyszerűsítésében, még mindig a tervezőkre támaszkodnak a végső döntések meghozatalában és a statikus felületek létrehozásában. A GenUI ezzel szemben azt jelenti, hogy az MI rendszer aktívabb szerepet vállal a felületek tényleges generálásában és adaptálásában a felhasználói adatok és kontextus alapján.

A GenUI jelentős változást jelent a felhasználói felület tervezésének megközelítésében, elmozdulva az univerzális megoldásoktól a nagymértékben személyre szabott, adaptív élmények felé. Az MI erejét kihasználva a GenUI forradalmasíthatja a digitális termékekkel és szolgáltatásokkal való interakciónkat, minden egyes felhasználó számára intuitívabb, lebilincselőbb és hatékonyabb felületeket létrehozva.

Példa

A GenUI koncepciójának szemléltetésére vegyünk egy hipotetikus fitness alkalmazást, a "FitAI"-t. Ez az alkalmazás személyre szabott edzésterveket és táplálkozási tanácsokat kínál a felhasználóknak egyéni céljaik, edzettségi szintjük és preferenciáik alapján.

Hagyományos UI-tervezési megközelítésben a FitAI-nak fix képernyőkészlete és elemei lennének, amelyek minden felhasználó számára ugyanazok. A GenUI-val azonban az alkalmazás felülete dinamikusan alkalmazkodhat minden felhasználó egyedi igényeihez és kontextusához.

Ez a megközelítés 2024-ben még meglehetősen merésznek tűnhet a megvalósítás szempontjából, és lehet, hogy még megfelelő megtérülést sem biztosítana, de lehetséges.

Íme, hogyan működhetne:

1. **Kezdeti beállítás**:

- Egy szabványos kérdőív helyett a FitAI beszélgető MI-t használ információgyűjtésre a felhasználó céljairól, jelenlegi edzettségi szintjéről és preferenciáiról.
- Ezen kezdeti interakció alapján az MI személyre szabott irányítópult-elrendezést generál, kiemelve a felhasználó céljai szempontjából legfontosabb funkciókat és információkat.
- A jelenlegi MI-technológia rendelkezésére állhat egy képernyőkomponens-készlet, amelyből összeállíthatja a személyre szabott irányítópultot.
- A jövőbeli MI-technológia átveheti egy tapasztalt UI-tervező szerepét, és ténylegesen *nullából* hozhatja létre az irányítópultot.

2. **Edzéstervező**:

- Az edzéstervező felületét a mesterséges intelligencia a felhasználó tapasztalati szintjéhez és a rendelkezésre álló eszközökhöz igazítja.
- Egy kezdő számára, akinek nincs felszerelése, egyszerű saját testsúlyos gyakorlatokat mutathat részletes útmutatókkal és videókkal.
- Egy haladó felhasználónak, aki edzőterembe jár, összetettebb edzésterveket jeleníthet meg kevesebb magyarázó tartalommal.
- Az edzéstervező tartalma nem egyszerűen egy nagy halmazból kerül szűrésre. A tartalom *menet közben* generálható egy olyan tudásbázis alapján, amelyet a felhasználóról ismert összes információval együtt kérdez le a rendszer.

3. **Fejlődéskövetés**:

- A fejlődéskövetési felület a felhasználó céljai és aktivitási mintái alapján fejlődik.
- Ha egy felhasználó elsősorban a fogyásra összpontosít, a felület kiemelten jelenítheti meg a súlyváltozás grafikonját és a kalóriaégetési statisztikákat.
- Egy izomépítő felhasználó esetében az erőnléti fejlődést és a testösszetétel változásait emelheti ki.

- A mesterséges intelligencia képes az alkalmazás ezen részét a felhasználó tényleges fejlődéséhez igazítani. Ha a fejlődés egy időre megtorpan, az alkalmazás olyan üzemmódba válthat, amelyben megpróbálja kideríteni a visszaesés okait, hogy segítsen azok leküzdésében.

4. **Táplálkozási Tanácsok**:

   - A táplálkozási szekció alkalmazkodik a felhasználó étkezési preferenciáihoz és korlátozásaihoz.
   - Egy vegán felhasználónak növényi alapú ételjavaslatokat és fehérjeforrásokat mutathat.
   - Egy gluténérzékeny felhasználó esetében automatikusan kiszűri a gluténtartalmú ételeket az ajánlásokból.
   - Itt is igaz, hogy a tartalom nem egy minden felhasználóra érvényes hatalmas ételadatbázisból származik, hanem egy olyan tudásbázisból kerül szintetizálásra, amely a felhasználó egyedi helyzetéhez és korlátozásaihoz igazítható információkat tartalmaz.
   - Például a receptek olyan összetevő-specifikációkkal generálódnak, amelyek megfelelnek a felhasználó folyamatosan változó kalóriaigényének, ahogy fittsége és testi mutatói fejlődnek.

5. **Motivációs Elemek**:

   - Az alkalmazás motivációs tartalma és értesítései személyre szabottak a felhasználó személyiségtípusa és a különböző motivációs stratégiákra adott reakciói alapján.
   - Egyes felhasználók bátorító üzeneteket kaphatnak, míg mások inkább adatvezérelt visszajelzéseket.

Ebben a példában a GenUI lehetővé teszi a FitAI számára, hogy minden felhasználó számára rendkívül személyre szabott élményt hozzon létre, potenciálisan növelve az elkötelezettséget, elégedettséget és a fitnesz célok elérésének valószínűségét. A felületi

elemek, a tartalom és még az alkalmazás "személyisége" is úgy alakul, hogy a legjobban szolgálja az egyes felhasználók igényeit és preferenciáit.

Az Átállás az Eredményközpontú Tervezésre

A GenUI alapvető változást jelent a felhasználói felület tervezésének megközelítésében, a konkrét felületi elemek létrehozásáról egy holisztikusabb, eredményközpontú megközelítésre való áttéréssel. Ennek a változásnak több fontos következménye van:

1. **Fókusz a Felhasználói Célokra**:
    - A tervezőknek mélyebben kell gondolkodniuk a felhasználói célokról és a kívánt eredményekről, nem pedig konkrét felületi komponensekről.
    - A hangsúly olyan rendszerek létrehozásán lesz, amelyek képesek olyan felületeket generálni, amelyek hatékonyan és eredményesen segítik a felhasználókat céljaik elérésében.
    - Új UI keretrendszerek fognak megjelenni, amelyek megadják az AI-alapú tervezőknek a szükséges eszközöket ahhoz, hogy *menet közben* és *nulláról* tudjanak felhasználói élményeket generálni, nem pedig előre meghatározott képernyő-specifikációk alapján.
2. **A Tervezők Változó Szerepe**:
    - A tervezők átállnak a fix elrendezések létrehozásáról a szabályok, korlátok és irányelvek meghatározására, amelyeket az AI rendszerek követnek a felületek generálásakor.
    - Új készségeket kell kifejleszteniük olyan területeken, mint az adatelemzés, az AI promptmérnökség és a rendszergondolkodás, hogy hatékonyan irányíthassák a GenUI rendszereket.
3. **A Felhasználói Kutatás Fontossága**:

- A felhasználói kutatás még kritikusabbá válik GenUI környezetben, mivel a tervezőknek nem csak a felhasználói preferenciákat kell megérteniük, hanem azt is, hogyan változnak ezek a preferenciák és igények különböző kontextusokban.
- A folyamatos felhasználói tesztelés és visszajelzési körök elengedhetetlenek lesznek az AI felületgenerálási képességeinek finomításához és fejlesztéséhez.

4. **Tervezés a Változatosságra**:

   - Ahelyett, hogy egyetlen "tökéletes" felületet hoznának létre, a tervezőknek több lehetséges variációt kell figyelembe venniük, és biztosítaniuk kell, hogy a rendszer megfelelő felületeket tudjon generálni a különböző felhasználói igényekhez.
   - Ez magában foglalja a szélsőséges esetek tervezését és annak biztosítását, hogy a generált felületek használhatósága és hozzáférhetősége megmaradjon a különböző konfigurációkban.
   - A termékek megkülönböztetése új dimenziókat nyer, amelyek magukban foglalják a felhasználói pszichológia eltérő perspektíváit és a versenytársak számára nem elérhető egyedi adathalmazok és tudásbázisok kihasználását.

Kihívások és Megfontolások

Bár a GenUI izgalmas lehetőségeket kínál, számos kihívást és megfontolnivalót is felvet:

1. **Technikai Korlátok**:

   - A jelenlegi AI technológia, bár fejlett, még mindig korlátozott a komplex felhasználói szándékok megértésében és a valóban kontextusérzékeny felületek generálásában.

- Teljesítményproblémák a felületi elemek valós idejű generálásával kapcsolatban, különösen a kevésbé erős eszközökön.

2. **Adatkövetelmények**:

   - A használati esettől függően a hatékony GenUI rendszerek jelentős mennyiségű felhasználói adatot igényelhetnek a személyre szabott felületek létrehozásához.
   - A hiteles felhasználói adatok etikus beszerzésének kihívásai adatvédelmi és biztonsági aggályokat vetnek fel, valamint potenciális torzításokat eredményezhetnek a GenUI modellek betanításához használt adatokban.

3. **Használhatóság és Következetesség**:

   - Legalábbis amíg ez a gyakorlat széles körben el nem terjed, az állandóan változó felületekkel rendelkező alkalmazás használhatósági problémákhoz vezethet, mivel a felhasználók nehezen találhatják meg az ismerős elemeket vagy navigálhatnak hatékonyan.
   - Kulcsfontosságú lesz az egyensúly megteremtése a személyre szabás és a következetes, tanulható felület fenntartása között.

4. **Túlzott AI-függőség**:

   - Fennáll a veszélye annak, hogy túlságosan az AI rendszerekre bízzuk a tervezési döntéseket, ami fantáziátlan, problémás vagy egyszerűen hibás felületi megoldásokhoz vezethet.
   - Az emberi felügyelet és az AI által generált tervezések felülbírálásának lehetősége a belátható jövőben is fontos marad.

5. **Akadálymentesítési Megfontolások**:

- A dinamikusan generált felületek fogyatékkal élő felhasználók számára való hozzáférhetőségének biztosítása teljesen új kihívásokat jelent, ami aggasztó, tekintve a tipikus rendszerek által mutatott gyenge szintű akadálymentesítési megfelelést.
- Másrészt az AI tervezők implementálhatók *beépített* akadálymentesítési szempontokkal, és képességekkel a hozzáférhető felületek menet közbeni létrehozására, ugyanúgy, ahogy a nem fogyatékkal élő felhasználók számára készítenek UI-t.
- Mindenképpen a GenUI rendszereket robusztus akadálymentesítési irányelvekkel és tesztelési folyamatokkal kell tervezni.

6. **Felhasználói Bizalom és Átláthatóság**:

- A felhasználók kellemetlenül érezhetik magukat olyan felületekkel kapcsolatban, amelyek "túl sokat tudnak" róluk vagy számukra érthetetlen módon változnak.
- Az átláthatóság biztosítása arról, hogy hogyan és miért személyre szabottak a felületek, fontos lesz a felhasználói bizalom kiépítéséhez.

Jövőkép és Lehetőségek

A Generatív UI (GenUI) jövője hatalmas ígéretet hordoz a digitális termékekkel és szolgáltatásokkal való interakciónk forradalmasítására. Ahogy ez a technológia tovább fejlődik, várhatóan alapvető változás következik be a felhasználói felületek tervezésében, megvalósításában és megtapasztalásában. Úgy gondolom, a GenUI az a jelenség, amely végre a tudományos-fantasztikus kategóriába tartozó szintre emeli szoftvereinket.

A GenUI egyik legizgalmasabb kilátása az akadálymentesítés olyan nagyságrendű fejlesztésének lehetősége, amely túlmutat azon, hogy egyszerűen biztosítsuk, hogy a súlyos fogyatékkal élő személyek ne legyenek teljesen kizárva a szoftver használatából.

A felületek automatikus alkalmazkodásával az egyéni felhasználói igényekhez, a GenUI minden eddiginél befogadóbbá teheti a digitális élményeket. Képzeljünk el olyan felületeket, amelyek zökkenőmentesen alkalmazkodnak nagyobb szövegmérettel a fiatalabb vagy látássérült felhasználók számára, vagy egyszerűsített elrendezést kínálnak a kognitív nehézségekkel küzdőknek, mindezt kézi konfiguráció vagy külön "akadálymentes" alkalmazásverziók nélkül.

A GenUI személyre szabási képességei várhatóan növelik a felhasználói elkötelezettséget, elégedettséget és lojalitást a digitális termékek széles körében. Ahogy a felületek jobban hangolódnak az egyéni preferenciákhoz és viselkedésekhez, a felhasználók intuitívabbnak és élvezetesebbnek találják majd a digitális élményeket, ami potenciálisan mélyebb és értelmesebb technológiai interakciókat eredményez.

A GenUI átalakíthatja az új felhasználók betanulási folyamatát is. Intuitív, személyre szabott első felhasználói élmények létrehozásával, amelyek gyorsan alkalmazkodnak minden felhasználó szakértelmi szintjéhez, a GenUI jelentősen csökkentheti az új alkalmazások tanulási görbéjét. Ez gyorsabb elfogadási arányokhoz és növekvő felhasználói magabiztossághoz vezethet az új funkciók és szolgáltatások felfedezésében.

Egy másik izgalmas lehetőség a GenUI képessége arra, hogy következetes felhasználói élményt biztosítson különböző eszközökön és platformokon, miközben optimalizál minden specifikus használati kontextusra. Ez megoldhatja a régóta fennálló kihívást, hogy koherens élményt nyújtsunk az egyre töredezettebbé váló eszközpalettán, az okostelefonoktól és táblagépektől kezdve az asztali számítógépeken át az olyan új technológiákig, mint a kiterjesztett valóságot megjelenítő szemüvegek.

A GenUI adatvezérelt természete lehetőségeket nyit a felhasználói felület tervezésének gyors iterációjára és fejlesztésére. A generált felületekkel való felhasználói interakciókról gyűjtött valós idejű adatok révén a tervezők és fejlesztők példátlan betekintést nyerhetnek a felhasználói viselkedésbe és preferenciákba. Ez a visszacsatolási hurok folyamatos fejlődéshez vezethet a UI tervezésben, amelyet a tényleges használati minták vezérelnek, nem pedig feltételezések vagy korlátozott felhasználói tesztelés.

E változásra való felkészüléshez a tervezőknek fejleszteniük kell készségeiket és szemléletmódjukat. A hangsúly a fix elrendezések létrehozásáról átkerül az átfogó tervezési rendszerek és irányelvek kidolgozására, amelyek információval láthatják el az AI-vezérelt felületgenerálást. A tervezőknek mélyreható ismereteket kell szerezniük az adatelemzés, az AI technológiák és a rendszergondolkodás terén, hogy hatékonyan irányíthassák a GenUI rendszereket.

Továbbá, ahogy a GenUI elmossa a határokat a tervezés és a technológia között, a tervezőknek szorosabban kell együttműködniük a fejlesztőkkel és adattudósokkal. Ez az interdiszciplináris megközelítés kulcsfontosságú lesz olyan GenUI rendszerek létrehozásában, amelyek nem csak vizuálisan vonzóak és felhasználóbarátok, hanem technikailag robusztusak és etikailag megalapozottak.

A GenUI etikai következményei is előtérbe kerülnek majd, ahogy a technológia fejlődik. A tervezők kulcsfontosságú szerepet játszanak majd a felelősségteljes mesterséges intelligencia használatának keretrendszerei kidolgozásában a felülettervezés során, biztosítva, hogy a személyre szabás javítsa a felhasználói élményt anélkül, hogy veszélyeztetné a magánszférát vagy etikátlan módon manipulálná a felhasználói viselkedést.

Ha a jövőbe tekintünk, a GenUI egyszerre kínál izgalmas lehetőségeket és jelentős kihívásokat. Megvan benne a potenciál, hogy intuitívabb, hatékonyabb és kielégítőbb digitális élményeket teremtsen a felhasználók számára világszerte. Bár a tervezőknek alkalmazkodniuk kell és új készségeket kell elsajátítaniuk, ugyanakkor példa nélküli lehetőséget kínál az ember-gép interakció jövőjének mélyreható és jelentőségteljes alakítására. A teljes körűen megvalósított GenUI rendszerek felé vezető út kétségtelenül összetett lesz, de a fejlettebb felhasználói élmény és a digitális hozzáférhetőség terén várható előnyök olyan jövőt ígérnek, amelyért érdemes küzdeni.

Intelligens munkafolyamat-vezénylés

Az alkalmazásfejlesztés területén a munkafolyamatok kulcsfontosságú szerepet játszanak a feladatok, folyamatok és felhasználói interakciók strukturálásában és végrehajtásában. Ahogy az alkalmazások egyre összetettebbé válnak és a felhasználói elvárások folyamatosan növekednek, az intelligens és adaptív munkafolyamat-vezénylés szükségessége egyre nyilvánvalóbbá válik.

Az "Intelligens munkafolyamat-vezénylési" megközelítés az AI komponensek kihasználására összpontosít, hogy dinamikusan vezényelje és optimalizálja az alkalmazásokon belüli komplex munkafolyamatokat. A cél olyan alkalmazások létrehozása, amelyek hatékonyabbak, reagálóképesebbek és alkalmazkodóbbak a valós idejű adatokhoz és kontextushoz.

Ebben a fejezetben feltárjuk az intelligens munkafolyamat-vezénylési megközelítés alapelveit és mintáit. Megvizsgáljuk, hogyan használható az AI a feladatok intelligens irányítására, a döntéshozatal automatizálására és a munkafolyamatok dinamikus adaptálására különböző tényezők, például a felhasználói viselkedés, rendszerteljesítmény és üzleti szabályok alapján. Gyakorlati példákon és valós forgatókönyveken keresztül bemutatjuk az AI átalakító potenciálját az alkalmazások munkafolyamatainak egyszerűsítésében és optimalizálásában.

Függetlenül attól, hogy összetett üzleti folyamatokkal rendelkező vállalati alkalmazásokat vagy dinamikus felhasználói utazásokkal rendelkező fogyasztói alkalmazásokat fejleszt, az ebben a fejezetben tárgyalt minták és technikák felruházzák Önt azzal a tudással és eszköztárral, amellyel intelligens és hatékony munkafolyamatokat hozhat létre, amelyek javítják a teljes felhasználói élményt és üzleti értéket teremtenek.

Üzleti igény

A munkafolyamat-kezelés hagyományos megközelítései gyakran előre meghatározott szabályokra és statikus döntési fákra támaszkodnak, amelyek merevek, rugalmatlanok lehetnek, és képtelenek megbirkózni a modern alkalmazások dinamikus természetével.

Vegyünk egy olyan forgatókönyvet, ahol egy e-kereskedelmi alkalmazásnak egy komplex megrendelés-teljesítési folyamatot kell kezelnie. A munkafolyamat több lépést is tartalmazhat, például megrendelés-ellenőrzést, készletellenőrzést, fizetésfeldolgozást, szállítást és ügyféltájékoztatást. Minden lépésnek megvan a saját szabályrendszere, függőségei, külső integrációi és kivételkezelési mechanizmusai. Egy ilyen munkafolyamat kézi vagy hardkódolt logikával történő kezelése gyorsan nehézkessé, hibára hajlamossá és nehezen karbantarthatóvá válhat.

Továbbá, ahogy az alkalmazás méreteződik és az egyidejű felhasználók száma növekszik, a munkafolyamatnak képesnek kell lennie alkalmazkodni és optimalizálni önmagát a

valós idejű adatok és rendszerteljesítmény alapján. Például csúcsforgalmi időszakokban az alkalmazásnak dinamikusan kell tudnia módosítani a munkafolyamatot bizonyos feladatok priorizálásához, az erőforrások hatékony elosztásához és a zökkenőmentes felhasználói élmény biztosításához.

Itt lép be az "Intelligens munkafolyamat-vezénylés" megközelítés. Az AI komponensek kihasználásával a fejlesztők olyan munkafolyamatokat hozhatnak létre, amelyek intelligensek, adaptívak és önoptimalizálók. Az AI képes hatalmas mennyiségű adat elemzésére, tanulni a múltbeli tapasztalatokból, és valós időben informált döntéseket hozni a munkafolyamat hatékony vezényléséhez.

Főbb előnyök

1. **Megnövekedett hatékonyság**: Az AI optimalizálhatja a feladatok elosztását, az erőforrás-kihasználtságot és a munkafolyamat végrehajtását, ami gyorsabb feldolgozási időt és jobb általános hatékonyságot eredményez.
2. **Alkalmazkodóképesség**: Az AI-vezérelt munkafolyamatok dinamikusan alkalmazkodhatnak a változó körülményekhez, például a felhasználói igények, rendszerteljesítmény vagy üzleti követelmények ingadozásaihoz, biztosítva, hogy az alkalmazás reagálóképes és rugalmas maradjon.
3. **Automatizált döntéshozatal**: Az AI automatizálhatja a komplex döntéshozatali folyamatokat a munkafolyamaton belül, csökkentve a manuális beavatkozást és minimalizálva az emberi hibák kockázatát.
4. **Személyre szabás**: Az AI elemezni tudja a felhasználói viselkedést, preferenciákat és kontextust a munkafolyamat személyre szabásához és egyéni felhasználói élmények nyújtásához.
5. **Skálázhatóság**: Az AI-vezérelt munkafolyamatok zökkenőmentesen skálázódhatnak a növekvő adatmennyiség és felhasználói interakciók kezelésére, a teljesítmény vagy megbízhatóság kompromittálása nélkül.

A következő szakaszokban feltárjuk azokat a kulcsfontosságú mintákat és technikákat, amelyek lehetővé teszik az intelligens munkafolyamatok megvalósítását, és bemutatjuk, hogyan alakítja át az AI a munkafolyamat-kezelést a modern alkalmazásokban.

Kulcsfontosságú minták

Az intelligens munkafolyamat-vezénylés alkalmazásokban történő megvalósításához a fejlesztők több kulcsfontosságú mintát használhatnak, amelyek kihasználják az AI erejét. Ezek a minták strukturált megközelítést nyújtanak a munkafolyamatok tervezéséhez és kezeléséhez, lehetővé téve az alkalmazások számára a folyamatok adaptálását, optimalizálását és automatizálását valós idejű adatok és kontextus alapján. Vizsgáljuk meg az intelligens munkafolyamat-vezénylés néhány alapvető mintáját.

Dinamikus feladatirányítás

Ez a minta az AI használatát jelenti a feladatok intelligens irányítására egy munkafolyamaton belül, különböző tényezők, például feladatprioritás, erőforrás-elérhetőség és rendszerteljesítmény alapján. Az AI algoritmusok elemezhetik minden feladat jellemzőit, figyelembe vehetik a rendszer aktuális állapotát, és informált döntéseket hozhatnak a feladatok legmegfelelőbb erőforrásokhoz vagy feldolgozási útvonalakhoz történő hozzárendeléséről. A dinamikus feladatirányítás biztosítja, hogy a feladatok hatékonyan legyenek elosztva és végrehajtva, optimalizálva a teljes munkafolyamat teljesítményét.

```
1  class TaskRouter
2    include Raix::ChatCompletion
3    include Raix::FunctionDispatch
4
5    attr_accessor :task
6
7    # list of functions that can be called by the AI entirely at its
8    # discretion depending on the task received
9
10   function :analyze_task_priority do
11     TaskPriorityAnalyzer.perform(task)
12   end
13
14   function :check_resource_availability, # ...
15   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
```

Figyeljük meg a 29. sorban található `while` kifejezés által létrehozott ciklust, amely folyamatosan kérdezi az AI-t, amíg a feladat kiosztásra nem kerül. A 35. sorban

elmentjük a feladat átiratát későbbi elemzés és hibakeresés céljából, ha szükségessé válik.

Kontextus Alapú Döntéshozatal

Nagyon hasonló kódot használhat a kontextus-tudatos döntések meghozatalához egy munkafolyamaton belül. A felhasználói preferenciák, történeti minták és valós idejű bemenetek elemzésével az AI komponensek meg tudják határozni a legmegfelelőbb cselekvési irányt a munkafolyamat minden egyes döntési pontjában. Igazítsa a munkafolyamat viselkedését minden felhasználó vagy forgatókönyv egyedi kontextusához, személyre szabott és optimalizált élményt nyújtva.

Adaptív Munkafolyamat-összeállítás

Ez a minta a munkafolyamatok dinamikus összeállítására és módosítására összpontosít a változó követelmények vagy feltételek alapján. Az AI elemzi a munkafolyamat aktuális állapotát, azonosítja a szűk keresztmetszeteket vagy hatékonytalanságokat, és automatikusan módosítja a munkafolyamat szerkezetét a teljesítmény optimalizálása érdekében. Az adaptív munkafolyamat-összeállítás lehetővé teszi, hogy az alkalmazások folyamatosan fejlődjenek és javítsák folyamataikat manuális beavatkozás nélkül.

Kivételkezelés és Helyreállítás

A kivételkezelés és helyreállítás kritikus szempontjai az intelligens munkafolyamat-vezérlésnek. AI komponensekkel és komplex munkafolyamatokkal való munka során elengedhetetlen, hogy előre lássuk és elegánsan kezeljük a kivételeket a rendszer stabilitásának és megbízhatóságának biztosítása érdekében.

Íme néhány kulcsfontosságú megfontolás és technika a kivételkezeléshez és helyreállításhoz az intelligens munkafolyamatokban:

1. **Kivétel Továbbítás**: Valósítson meg egy következetes megközelítést a kivételek továbbítására a munkafolyamat komponensek között. Amikor kivétel történik egy komponensen belül, azt el kell kapni, naplózni kell, és továbbítani kell az vezérlőhöz vagy egy különálló komponenshez, amely felelős a kivételek kezeléséért. A cél a kivételkezelés központosítása és annak megakadályozása, hogy a kivételek csendben elnyelődjenek, valamint lehetőségek nyitása az Intelligens Hibakezelés számára.
2. **Újrapróbálkozási Mechanizmusok**: Az újrapróbálkozási mechanizmusok segítenek javítani a munkafolyamat ellenálló képességét és kezelni az átmeneti hibákat. Mindenképpen próbáljon meg újrapróbálkozási mechanizmusokat implementálni az átmeneti vagy helyreállítható kivételekhez, mint például a hálózati kapcsolat vagy erőforrás-elérhetetlenség, amelyek automatikusan újrapróbálhatók egy meghatározott késleltetés után. Egy AI-vezérelt vezérlő vagy kivételkezelő azt jelenti, hogy az újrapróbálkozási stratégiáknak nem kell mechanikusnak lenniük, fix algoritmusokra támaszkodva, mint az exponenciális visszalépés. A kivétel kezelésének módját rábízhatja a kivétel kezeléséért felelős AI komponens "belátására".
3. **Tartalék Stratégiák**: Ha egy AI komponens nem tud érvényes választ adni vagy hibába ütközik - ami gyakran előfordul a legmodernebb technológiák esetében - legyen készenlétben egy tartalék mechanizmus, hogy a munkafolyamat folytatódhasson. Ez lehet alapértelmezett értékek használata, alternatív algoritmusok, vagy egy Ember a Hurokban a döntések meghozatalához és a munkafolyamat előreviteléhez.
4. **Kompenzációs Műveletek**: A vezérlő utasításainak tartalmazniuk kell a kompenzációs műveletekre vonatkozó utasításokat az automatikusan nem megoldható kivételek kezelésére. A kompenzációs műveletek olyan lépések, amelyeket egy sikertelen művelet hatásainak visszavonására vagy enyhítésére tesznek. Például, ha egy fizetési feldolgozási lépés sikertelen, a kompenzációs művelet lehet a tranzakció visszagörgetése és a felhasználó értesítése. A

kompenzációs műveletek segítenek fenntartani az adatok konzisztenciáját és integritását kivételek esetén.

5. **Kivétel Monitorozás és Riasztás**: Állítson fel monitorozási és riasztási mechanizmusokat a kritikus kivételek észlelésére és az érintett érdekeltek értesítésére. A vezérlő tudatában lehet a küszöbértékeknek és szabályoknak, hogy riasztásokat indítson, amikor a kivételek meghaladnak bizonyos határértékeket, vagy amikor bizonyos típusú kivételek fordulnak elő. Ez lehetővé teszi a problémák proaktív azonosítását és megoldását, mielőtt azok hatással lennének a teljes rendszerre.

Íme egy példa a kivételkezelésre és helyreállításra egy Ruby munkafolyamat komponensben:

```
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
```

Ebben a példában az `InventoryManager` komponens ellenőrzi egy termék elérhetőségét egy adott megrendeléshez. Ha a rendelkezésre álló mennyiség nem elegendő, egy `InsufficientInventoryError` kivételt dob. A kivételt a rendszer elkapja, naplózza, és újrapróbálkozási mechanizmust alkalmaz. Ha az újrapróbálkozások száma meghaladja a limitet, a komponens manuális beavatkozásra vált át egy adminisztrátor értesítésével.

A robosztus kivételkezelési és helyreállítási mechanizmusok implementálásával biztosíthatjuk, hogy az intelligens munkafolyamataink ellenállóak, karbantarthatóak és képesek elegánsan kezelni a váratlan helyzeteket.

Ezek a minták képezik az intelligens munkafolyamat-vezérlés alapját, és különböző alkalmazások specifikus követelményeihez kombinálhatók és adaptálhatók. Ezeket a mintákat alkalmazva a fejlesztők olyan munkafolyamatokat hozhatnak létre, amelyek rugalmasak, ellenállóak és optimalizáltak mind a teljesítmény, mind a felhasználói élmény szempontjából.

A következő részben azt vizsgáljuk meg, hogyan lehet ezeket a mintákat a gyakorlatban megvalósítani, valós példákon és kódrészleteken keresztül szemléltetve az AI komponensek integrációját a munkafolyamat-kezelésbe.

Intelligens munkafolyamat-vezérlés megvalósítása a gyakorlatban

Most, hogy megismertük az intelligens munkafolyamat-vezérlés kulcsfontosságú mintáit, nézzük meg, hogyan lehet ezeket valós alkalmazásokban implementálni. Gyakorlati példákat és kódrészleteket mutatunk be az AI komponensek munkafolyamat-kezelésbe történő integrációjának szemléltetésére.

Intelligens Megrendelés-feldolgozó

Nézzünk meg egy gyakorlati példát az intelligens munkafolyamat-vezérlés megvalósítására egy AI-alapú `OrderProcessor` komponens segítségével egy Ruby on Rails e-kereskedelmi alkalmazásban. Az `OrderProcessor` megvalósítja a Folyamatkezelő Vállalati Integráció koncepciót, amellyel először a 3. fejezetben találkoztunk a Munkavégzők Sokasága tárgyalásakor. A komponens felelős a megrendelések teljesítési folyamatának kezeléséért, az időközi eredmények alapján történő útválasztási döntések meghozataláért és a különböző feldolgozási lépések végrehajtásának vezérléséért.

A megrendelés teljesítési folyamata több lépésből áll, mint például a megrendelés érvényesítése, készletellenőrzés, fizetés feldolgozása és szállítás. Minden lépést külön munkavégző folyamat valósít meg, amely egy adott feladatot hajt végre és visszaadja az eredményt az `OrderProcessor`-nak. A lépések nem kötelezőek, és nem is feltétlenül kell őket pontos sorrendben végrehajtani.

Itt látható egy példa az `OrderProcessor` implementációjára. Két mixint tartalmaz a Raix könyvtárból. Az első (`ChatCompletion`) chat completion képességet ad neki, ami AI komponenssé teszi. A második (`FunctionDispatch`) lehetővé teszi az AI számára a függvényhívást, így egy promptra nem szöveges üzenettel, hanem függvényhívással tud válaszolni.

A munkavégző függvények (`validate_order`, `check_inventory`, stb.) a megfelelő munkavégző osztályokhoz delegálják a feladatokat, amelyek lehetnek AI vagy nem-AI komponensek, azzal az egyetlen követelménnyel, hogy a munkájuk eredményét olyan formátumban adják vissza, amely szövegként ábrázolható.

Mint a könyv ezen részének minden más példája esetében, ez a kód gyakorlatilag pszeudokód, és csak a minta jelentését hivatott közvetíteni, valamint inspirálni saját alkotásaidat. A minták teljes leírása és a teljes kódpéldák a 2. részben találhatók.

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

  SYSTEM_DIRECTIVE = "You are an order processor, tasked with..."

  def initialize(order)
    self.order = order
    transcript << { system: SYSTEM_DIRECTIVE }
    transcript << { user: order.to_json }
  end

  def perform
    # will continue looping until `stop_looping!` is called
    chat_completion(loop: true)
  end

  # list of functions available to be called by the AI
  # truncated for brevity

  def functions
    [
      {
        name: "validate_order",
        description: "Invoke to check validity of order",
        parameters: {
          ...
      },
```

```
      ...
    ]
  end

  # implementation of functions that can be called by the AI
  # entirely at its discretion, depending on the needs of the order

  def validate_order
    OrderValidationWorker.perform(@order)
  end

  def check_inventory
    InventoryCheckWorker.perform(@order)
  end

  def process_payment
    PaymentProcessingWorker.perform(@order)
  end

  def schedule_shipping
    ShippingSchedulerWorker.perform(@order)
  end

  def send_confirmation
    OrderConfirmationWorker.perform(@order)
  end

  def finished_processing
    @order.update!(transcript:, processed_at: Time.current)
    stop_looping!
  end
end
```

A példában az OrderProcessor egy rendelési objektummal kerül inicializálásra, és fenntartja a munkafolyamat végrehajtásának átiratát, a nagy nyelvi modellek sajátos párbeszédes átirat formátumában. A mesterséges intelligencia teljes körű irányítást kap a különböző feldolgozási lépések végrehajtásának összehangolására, mint például a rendelés érvényesítése, készletellenőrzés, fizetés feldolgozása és szállítás.

Minden alkalommal, amikor a `chat_completion` metódus meghívásra kerül, az átirat

elküldésre kerül az AI-nak, hogy az egy függvényhívás formájában adjon választ. Teljesen az AI-n múlik, hogy elemezze az előző lépés eredményét és meghatározza a megfelelő következő lépést. Például, ha a készletellenőrzés alacsony készletszintet mutat, az `OrderProcessor` ütemezhet egy feltöltési feladatot. Ha a fizetés feldolgozása sikertelen, kezdeményezhet egy újrapróbálkozást vagy értesítheti az ügyfélszolgálatot.

> A fenti példában nincsenek definiálva függvények a feltöltéshez vagy az ügyfélszolgálat értesítéséhez, de természetesen lehetnének.

Az átirat minden függvényhívással bővül, és a munkafolyamat végrehajtásának nyilvántartásaként szolgál, beleértve az egyes lépések eredményeit és az AI által generált utasításokat a következő lépésekhez. Ez az átirat felhasználható hibakeresésre, auditálásra és a rendeléstelje sítési folyamat átláthatóságának biztosítására.

Az AI `OrderProcessor`-ban történő alkalmazásával az e-kereskedelmi alkalmazás dinamikusan tudja adaptálni a munkafolyamatot a valós idejű adatok alapján, és intelligensen tudja kezelni a kivételeket. Az AI komponens képes informált döntéseket hozni, optimalizálni a munkafolyamatot, és biztosítani a zökkenőmentes rendelésfeldolgozást még összetett helyzetekben is.

Az a tény, hogy a feldolgozó folyamatok egyetlen követelménye, hogy valamilyen értelmezhető kimenetet adjanak az AI számára a következő lépés meghatározásához, ráébreszthet arra, hogy ez a megközelítés hogyan csökkentheti a különböző rendszerek integrációjánál általában szükséges bemenet/kimenet leképezési munkát.

Intelligens Tartalommoderátor

A közösségi média alkalmazások általában legalább minimális tartalommoderációt igényelnek a biztonságos és egészséges közösség biztosításához. Ez a példa `ContentModerator` komponens az AI-t használja fel a moderációs munkafolyamat

intelligens összehangolására, döntéseket hozva a tartalom jellemzői és a különböző moderációs lépések eredményei alapján.

A moderációs folyamat több lépést foglal magában, mint például szövegelemzést, képfelismerést, felhasználói hírnév értékelését és kézi áttekintést. Minden lépés külön feldolgozó folyamatként van megvalósítva, amely elvégez egy specifikus feladatot és visszaadja az eredményt a `ContentModerator`-nak.

Íme egy példa implementáció a `ContentModerator`-ról:

```
class ContentModerator
  include Raix::ChatCompletion
  include Raix::FunctionDispatch

  SYSTEM_DIRECTIVE = "You are a content moderator process manager,
    tasked with the workflow involved in moderating user-generated content..."

  def initialize(content)
    @content = content
    @transcript = [
      { system: SYSTEM_DIRECTIVE },
      { user: content.to_json }
    ]
  end

  def perform
    complete(@transcript)
  end

  def model
    "openai/gpt-4"
  end

  # list of functions available to be called by the AI
  # truncated for brevity

  def functions
    [
      {
        name: "analyze_text",
```

```
        # ...
      },
      {
        name: "recognize_image",
        description: "Invoke to describe images...",
        # ...
      },
      {
        name: "assess_user_reputation",
        # ...
      },
      {
        name: "escalate_to_manual_review",
        # ...
      },
      {
        name: "approve_content",
        # ...
      },
      {
        name: "reject_content",
        # ...
      }
    ]
  end

  # implementation of functions that can be called by the AI
  # entirely at its discretion, depending on the needs of the order

  def analyze_text
    result = TextAnalysisWorker.perform(@content)
    continue_with(result)
  end

  def recognize_image
    result = ImageRecognitionWorker.perform(@content)
    continue_with(result)
  end

  def assess_user_reputation
    result = UserReputationWorker.perform(@content.user)
    continue_with(result)
```

```
  end

  def escalate_to_manual_review
    ManualReviewWorker.perform(@content)
    @content.update!(status: 'pending', transcript: @transcript)
  end

  def approve_content
    @content.update!(status: 'approved', transcript: @transcript)
  end

  def reject_content
    @content.update!(status: 'rejected', transcript: @transcript)
  end

  private

  def continue_with(result)
    @transcript << { function: result }
    complete(@transcript)
  end
end
```

Ebben a példában a `ContentModerator` egy tartalmi objektummal kerül inicializálásra, és egy moderálási jegyzőkönyvet vezet beszélgetés formátumban. Az AI komponens teljes kontrollt gyakorol a moderálási munkafolyamat felett, és a tartalom jellemzői, valamint az egyes lépések eredményei alapján dönti el, mely lépéseket hajtsa végre.

Az AI által meghívható munkafüggvények közé tartozik az `analyze_text`, `recognize_image`, `assess_user_reputation` és az `escalate_to_manual_review`. Minden függvény a megfelelő munkafolyamathoz (`TextAnalysisWorker`, `ImageRecognitionWorker`, stb.) delegálja a feladatot, és az eredményt hozzáfűzi a moderálási jegyzőkönyvhöz, kivéve az eszkalációs függvényt, amely végállapotként működik. Végül, az `approve_content` és `reject_content` függvények szintén végállapotként szolgálnak.

Az AI komponens elemzi a tartalmat és meghatározza a megfelelő intézkedést. Ha a tartalom képhivatkozásokat tartalmaz, meghívhatja a `recognize_image`

munkafüggvényt a vizuális ellenőrzés támogatásához. Ha bármely munkafüggvény potenciálisan káros tartalomra figyelmeztet, az AI dönthet úgy, hogy eszkalálja a tartalmat kézi felülvizsgálatra, vagy egyszerűen elutasítja azt. A figyelmeztetés súlyosságától függően azonban az AI úgy is dönthet, hogy a felhasználói hírnév értékelésének eredményeit használja fel az olyan tartalmak kezelésében, amelyekben egyébként nem biztos. Az alkalmazási esettől függően előfordulhat, hogy a megbízható felhasználók több mozgásteret kapnak azzal kapcsolatban, hogy mit tehetnek közzé. És így tovább…

Akárcsak az előző folyamatkezelő példában, a moderálási jegyzőkönyv a munkafolyamat végrehajtásának nyilvántartásaként szolgál, beleértve az egyes lépések eredményeit és az AI által generált döntéseket. Ez a jegyzőkönyv felhasználható ellenőrzésre, átláthatóságra és a moderálási folyamat idővel történő fejlesztésére.

A `ContentModerator` AI-alapú megközelítésével a közösségi média alkalmazás dinamikusan adaptálhatja a moderálási munkafolyamatot a tartalom jellemzői alapján, és intelligensen kezelheti a komplex moderálási forgatókönyveket. Az AI komponens képes megalapozott döntéseket hozni, optimalizálni a munkafolyamatot és biztonságos, egészséges közösségi élményt biztosítani.

Nézzünk meg két további példát, amelyek bemutatják a prediktív feladatütemezést, valamint a kivételkezelést és helyreállítást az intelligens munkafolyamat-vezérlés kontextusában.

Prediktív Feladatütemezés egy Ügyfélszolgálati Rendszerben

Egy Ruby on Rails alapú ügyfélszolgálati alkalmazásban kulcsfontosságú a támogatási jegyek hatékony kezelése és priorizálása az ügyfelek időben történő segítségnyújtása érdekében. A `SupportTicketScheduler` komponens AI-t használ a támogatási jegyek prediktív ütemezésére és hozzárendelésére az elérhető ügyintézőkhöz,

különböző tényezők alapján, mint például a jegy sürgőssége, az ügyintézői szaktudás és a munkaterhelés.

```
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 }
```

```
82      complete(@transcript)
83    end
84  end
```

Ebben a példában a `SupportTicketScheduler` egy támogatási jegy objektummal kerül inicializálásra és egy ütemezési jegyzőkönyvet vezet. Az AI komponens elemzi a jegy részleteit, és prediktív módon ütemezi a jegy hozzárendelését olyan tényezők alapján, mint a jegy sürgőssége, az ügyintézői szaktudás és az előrejelzett ügyintézői munkaterhelés.

Az AI által meghívható funkciók közé tartozik az `analyze_ticket_urgency`, `list_available_agents`, `predict_agent_workload` és az `assign_ticket_to_agent`. Minden függvény a megfelelő elemző vagy előrejelző komponenshez delegálja a feladatot, és az eredményt hozzáfűzi az ütemezési jegyzőkönyvhöz. Az AI-nak lehetősége van a hozzárendelés késleltetésére is a `delay_assignment` függvény használatával.

Az AI komponens megvizsgálja az ütemezési jegyzőkönyvet és megalapozott döntéseket hoz a jegyek kiosztásáról. Figyelembe veszi a jegy sürgősségét, az elérhető ügyintézők szaktudását és minden ügyintéző előrejelzett munkaterhelését annak meghatározásához, hogy melyik ügyintéző a legalkalmasabb a jegy kezelésére.

A prediktív feladatütemezés kihasználásával az ügyfélszolgálati alkalmazás optimalizálhatja a jegyek kiosztását, csökkentheti a válaszidőket és javíthatja az általános ügyfélelégedettséget. A támogatási jegyek proaktív és hatékony kezelése biztosítja, hogy a megfelelő jegyek a megfelelő ügyintézőkhöz kerüljenek a megfelelő időben.

Kivételkezelés és helyreállítás az adatfeldolgozási folyamatban

A kivételek kezelése és a hibákból való helyreállás elengedhetetlen az adatintegritás biztosításához és az adatvesztés megelőzéséhez. A `DataProcessingOrchestrator`

komponens mesterséges intelligenciát használ a kivételek intelligens kezeléséhez és a helyreállítási folyamat irányításához az adatfeldolgozási folyamatban

```
class DataProcessingOrchestrator
  include Raix::ChatCompletion
  include Raix::FunctionDispatch

  SYSTEM_DIRECTIVE = "You are a data processing orchestrator..."

  def initialize(data_batch)
    @data_batch = data_batch
    @transcript = [
      { system: SYSTEM_DIRECTIVE },
      { user: data_batch.to_json }
    ]
  end

  def perform
    complete(@transcript)
  end

  def model
    "openai/gpt-4"
  end

  def functions
    [
      {
        name: "validate_data",
        # ...
      },
      {
        name: "process_data",
        # ...
      },
      {
        name: "request_fix",
        # ...
      },
      {
        name: "retry_processing",
        # ...
```

```
      },
      {
        name: "mark_data_as_failed",
        # ...
      },
      {
        name: "finished",
        # ...
      }
    ]
  end

  # implementation of functions that can be called by the AI
  # entirely at its discretion, depending on the needs of the order

  def validate_data
    result = DataValidator.perform(@data_batch)
    continue_with(result)
  rescue ValidationException => e
    handle_validation_exception(e)
  end

  def process_data
    result = DataProcessor.perform(@data_batch)
    continue_with(result)
  rescue ProcessingException => e
    handle_processing_exception(e)
  end

  def request_fix(description_of_fix)
    result = SmartDataFixer.new(description_of_fix, @data_batch)
    continue_with(result)
  end

  def retry_processing(timeout_in_seconds)
    wait(timeout_in_seconds)
    process_data
  end

  def mark_data_as_failed
    @data_batch.update!(status: 'failed', transcript: @transcript)
  end
```

```
82
83    def finished
84      @data_batch.update!(status: 'finished', transcript: @transcript)
85    end
86
87    private
88
89    def continue_with(result)
90      @transcript << { function: result }
91      complete(@transcript)
92    end
93
94    def handle_validation_exception(exception)
95      @transcript << { exception: exception.message }
96      complete(@transcript)
97    end
98
99    def handle_processing_exception(exception)
100     @transcript << { exception: exception.message }
101     complete(@transcript)
102   end
103 end
```

Ebben a példában a `DataProcessingOrchestrator` egy adatköteg objektummal inicializálódik és fenntart egy feldolgozási jegyzőkönyvet. Az AI komponens vezényli az adatfeldolgozási folyamatot, kezeli a kivételeket és szükség esetén helyreállítja a hibákat.

Az AI által meghívható funkciók közé tartozik a `validate_data`, `process_data`, `request_fix`, `retry_processing` és a `mark_data_as_failed`. Minden függvény delegálja a feladatot egy megfelelő adatfeldolgozó komponensnek, és hozzáfűzi az eredményt vagy a kivétel részleteit a feldolgozási jegyzőkönyvhöz.

Ha validációs kivétel történik a `validate_data` lépés során, a `handle_validation_exception` függvény hozzáfűzi a kivétel adatait a jegyzőkönyvhöz, és visszaadja az irányítást az AI-nak. Hasonlóképpen, ha feldolgozási kivétel történik a `process_data` lépés során, az AI dönthet a helyreállítási stratégiáról.

A fellépő kivétel természetétől függően az AI saját belátása szerint dönthet úgy,

hogy meghívja a `request_fix` függvényt, amely egy AI-vezérelt `SmartDataFixer` komponenshez delegál (lásd az Öngyógyító Adatok fejezetet). Az adatjavító egy egyszerű angol nyelvű leírást kap arról, hogyan kell módosítania a `@data_batch`-et, hogy a feldolgozást újra lehessen próbálni. Talán egy sikeres újrapróbálkozás magában foglalná a validációt nem teljesítő rekordok eltávolítását az adatkötegből és/vagy azok átmásolását egy másik feldolgozási folyamatba emberi felülvizsgálatra? A lehetőségek szinte végtelenek.

Az AI-vezérelt kivételkezelés és helyreállítás beépítésével az adatfeldolgozó alkalmazás ellenállóbbá és hibatűrőbbé válik. A `DataProcessingOrchestrator` intelligensen kezeli a kivételeket, minimalizálja az adatvesztést és biztosítja az adatfeldolgozási munkafolyamat zökkenőmentes végrehajtását.

Monitorozás és Naplózás

A monitorozás és naplózás láthatóságot biztosít az AI-vezérelt munkafolyamat-komponensek előrehaladásába, teljesítményébe és állapotába, lehetővé téve a fejlesztők számára a rendszer viselkedésének nyomon követését és elemzését. A hatékony monitorozási és naplózási mechanizmusok megvalósítása elengedhetetlen az intelligens munkafolyamatok hibakeresése, ellenőrzése és folyamatos fejlesztése szempontjából.

Munkafolyamat Előrehaladásának és Teljesítményének Monitorozása

Az intelligens munkafolyamatok zökkenőmentes végrehajtásának biztosításához fontos monitorozni minden munkafolyamat-komponens előrehaladását és teljesítményét. Ez magában foglalja a kulcsfontosságú mérőszámok és események nyomon követését a munkafolyamat teljes életciklusa során.

Néhány fontos monitorozandó szempont:

1. **Munkafolyamat Végrehajtási Ideje**: Mérjük az egyes munkafolyamat-komponensek feladatainak végrehajtásához szükséges időt. Ez segít azonosítani a teljesítmény szűk keresztmetszeteit és optimalizálni a teljes munkafolyamat hatékonyságát.

2. **Erőforrás-kihasználtság**: Figyeljük az egyes munkafolyamat-komponensek rendszererőforrás-használatát, mint például a CPU, memória és tárhely kihasználtságát. Ez segít biztosítani, hogy a rendszer kapacitásán belül működjön és hatékonyan kezelje a munkaterhelést.

3. **Hibaarányok és Kivételek**: Kövessük nyomon a hibák és kivételek előfordulását a munkafolyamat-komponenseken belül. Ez segít azonosítani a potenciális problémákat és lehetővé teszi a proaktív hibakezelést és helyreállítást.

4. **Döntési Pontok és Eredmények**: Monitorozzuk a munkafolyamaton belüli döntési pontokat és az AI-vezérelt döntések kimeneteleit. Ez betekintést nyújt az AI komponensek viselkedésébe és hatékonyságába.

A monitorozási folyamatok által gyűjtött adatok megjeleníthetők irányítópultokon vagy felhasználhatók ütemezett jelentések bemeneteként, amelyek tájékoztatják a rendszeradminisztrátorokat a rendszer állapotáról.

A monitorozási adatokat be lehet táplálni egy AI-vezérelt rendszeradminisztrátori folyamatba áttekintésre és esetleges intézkedésre!

Kulcsfontosságú Események és Döntések Naplózása

A naplózás egy elengedhetetlen gyakorlat, amely magában foglalja a munkafolyamat végrehajtása során előforduló kulcsfontosságú események, döntések és kivételek rögzítését és tárolását.

Néhány fontos naplózandó szempont:

1. **Munkafolyamat Indítása és Befejezése**: Naplózzuk minden munkafolyamat-példány kezdési és befejezési időpontját, valamint minden releváns metaadatot, mint például a bemeneti adatokat és felhasználói kontextust.

2. **Komponens Végrehajtás**: Naplózzuk minden munkafolyamat-komponens végrehajtási részleteit, beleértve a bemeneti paramétereket, kimeneti eredményeket és minden generált köztes adatot.

3. **AI Döntések és Indoklás**: Naplózzuk az AI komponensek által hozott döntéseket, valamint az alapul szolgáló indoklást vagy megbízhatósági pontszámokat. Ez átláthatóságot biztosít és lehetővé teszi az AI-vezérelt döntések ellenőrzését.

4. **Kivételek és Hibaüzenetek**: Naplózzuk a munkafolyamat végrehajtása során fellépő kivételeket vagy hibaüzeneteket, beleértve a veremkivonatot és a releváns kontextusinformációkat.

A naplózás különböző technikákkal valósítható meg, például naplófájlokba írással, naplók adatbázisban tárolásával vagy naplók központosított naplózási szolgáltatásba küldésével. Fontos olyan naplózási keretrendszert választani, amely rugalmasságot, skálázhatóságot és könnyű integrációt biztosít az alkalmazás architektúrájával.

Íme egy példa arra, hogyan lehet megvalósítani a naplózást egy Ruby on Rails alkalmazásban az `ActiveSupport::Logger` osztály használatával:

```ruby
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)
```

A munkafolyamat-komponensekben és a mesterséges intelligencia döntési pontjaiban stratégiailag elhelyezett naplózási utasításokkal a fejlesztők értékes információkat gyűjthetnek hibakereséshez, auditáláshoz és elemzéshez.

A Megfigyelés és Naplózás Előnyei

Az intelligens munkafolyamat-vezérlésben a megfigyelés és naplózás bevezetése számos előnnyel jár:

1. **Hibakeresés és Hibaelhárítás**: A részletes naplók és megfigyelési adatok segítik a fejlesztőket a problémák gyors azonosításában és diagnosztizálásában. Betekintést nyújtanak a munkafolyamat végrehajtási menetébe, a komponensek közötti interakciókba és az esetlegesen felmerülő hibákba vagy kivételekbe.

2. **Teljesítményoptimalizálás**: A teljesítménymutatók megfigyelése lehetővé teszi a fejlesztők számára a szűk keresztmetszetek azonosítását és a munkafolyamat-komponensek optimalizálását a jobb hatékonyság érdekében. A végrehajtási idők, erőforrás-kihasználtság és egyéb mutatók elemzésével a fejlesztők megalapozott döntéseket hozhatnak a rendszer általános teljesítményének javítása érdekében.

3. **Auditálás és Megfelelőség**: A kulcsfontosságú események és döntések naplózása biztosítja az auditálási nyomvonalat a szabályozási megfelelőség és elszámoltathatóság érdekében. Lehetővé teszi a szervezetek számára a mesterséges intelligencia komponensek által végrehajtott műveletek nyomon követését és ellenőrzését, valamint az üzleti szabályoknak és jogi követelményeknek való megfelelés biztosítását.

4. **Folyamatos Fejlesztés**: A megfigyelési és naplózási adatok értékes inputként szolgálnak az intelligens munkafolyamatok folyamatos fejlesztéséhez. A történeti adatok elemzésével, minták azonosításával és a mesterséges intelligencia döntések hatékonyságának mérésével a fejlesztők iteratív módon finomíthatják és fejleszthetik a munkafolyamat-vezérlési logikát.

Megfontolások és Legjobb Gyakorlatok

Az intelligens munkafolyamat-vezérlésben a megfigyelés és naplózás megvalósításakor vegye figyelembe a következő legjobb gyakorlatokat:

1. **Egyértelmű Megfigyelési Mutatók Meghatározása**: Azonosítsa a munkafolyamat specifikus követelményei alapján megfigyelendő kulcsfontosságú mutatókat és eseményeket. Összpontosítson azokra a mutatókra, amelyek értelmes betekintést nyújtanak a rendszer teljesítményébe, állapotába és viselkedésébe.

2. **Részletes Naplózás Megvalósítása**: Gondoskodjon róla, hogy a naplózási utasítások megfelelő helyen legyenek elhelyezve a munkafolyamat-komponensekben és a mesterséges intelligencia döntési pontjaiban. Rögzítse a releváns kontextusinformációkat, például a bemeneti paramétereket, kimeneti eredményeket és minden generált köztes adatot.

3. **Strukturált Naplózás Használata**: Alkalmazzon strukturált naplózási formátumot a naplóadatok könnyű elemzése és feldolgozása érdekében. A strukturált naplózás jobb kereshetőséget, szűrést és a naplóbejegyzések összesítését teszi lehetővé.

4. **Napló Megőrzés és Rotáció Kezelése**: Vezessen be napló megőrzési és rotációs szabályzatokat a naplófájlok tárolásának és életciklusának kezelésére. Határozza meg a megfelelő megőrzési időszakot a jogi követelmények, tárolási korlátok és elemzési igények alapján. Ha lehetséges, szervezze ki a naplózást olyan harmadik féltől származó szolgáltatásokhoz, mint a Papertrail.

5. **Érzékeny Információk Védelme**: Legyen körültekintő az érzékeny információk naplózásakor, mint például a személyazonosításra alkalmas információk (PII) vagy bizalmas üzleti adatok. Alkalmazzon megfelelő biztonsági intézkedéseket, például adatmaszkolást vagy titkosítást az érzékeny információk védelmére a naplófájlokban.

6. **Integrálás Megfigyelési és Riasztási Eszközökkel**: Használjon megfigyelési és riasztási eszközöket a megfigyelési és naplózási adatok központosított gyűjtéséhez, elemzéséhez és megjelenítéséhez. Ezek az eszközök valós idejű betekintést nyújthatnak,

előre meghatározott küszöbértékek alapján riasztásokat generálhatnak, és elősegíthetik a proaktív problémafeltárást és -megoldást. Az én kedvenc eszközöm ezek közül a Datadog.

Az átfogó megfigyelési és naplózási mechanizmusok megvalósításával a fejlesztők értékes betekintést nyerhetnek az intelligens munkafolyamatok viselkedésébe és teljesítményébe. Ezek a betekintések lehetővé teszik a mesterséges intelligencia által támogatott munkafolyamat-vezérlő rendszerek hatékony hibakeresését, optimalizálását és folyamatos fejlesztését.

Skálázhatósági és Teljesítmény Megfontolások

A skálázhatóság és a teljesítmény kritikus szempontok az intelligens munkafolyamat-vezérlő rendszerek tervezésekor és megvalósításakor. Ahogy az egyidejű munkafolyamatok mennyisége és a mesterséges intelligencia által támogatott komponensek összetettsége növekszik, elengedhetetlenné válik annak biztosítása, hogy a rendszer hatékonyan tudja kezelni a munkaterhelést és zökkenőmentesen tudjon skálázódni a növekvő igények kielégítésére.

Nagy Mennyiségű Egyidejű Munkafolyamat Kezelése

Az intelligens munkafolyamat-vezérlő rendszereknek gyakran nagy számú egyidejű munkafolyamatot kell kezelniük. A skálázhatóság biztosítása érdekében vegye figyelembe a következő stratégiákat:

1. **Aszinkron Feldolgozás**: Valósítson meg aszinkron feldolgozási mechanizmusokat a munkafolyamat-komponensek végrehajtásának szétválasztására. Ez lehetővé teszi a rendszer számára több munkafolyamat egyidejű kezelését anélkül, hogy blokkolná vagy várnia kellene az egyes komponensek befejezésére. Az aszinkron feldolgozás megvalósítható üzenetsorok, eseményvezérelt architektúrák vagy olyan háttérfeladat-feldolgozó keretrendszerek használatával, mint a Sidekiq.

2. **Elosztott Architektúra**: Tervezze úgy a rendszerarchitektúrát, hogy szervermentesen működő komponenseket (például AWS Lambda) használjon, vagy egyszerűen ossza el a munkaterhelést több csomópont vagy szerver között a fő alkalmazásszerver mellett. Ez lehetővé teszi a horizontális skálázhatóságot, ahol további csomópontok adhatók hozzá a megnövekedett munkafolyamat-mennyiség kezelésére.

3. **Párhuzamos Végrehajtás**: Azonosítsa a párhuzamos végrehajtás lehetőségeit a munkafolyamatokon belül. Egyes munkafolyamat-komponensek függetlenek lehetnek egymástól és párhuzamosan végrehajthatók. A párhuzamos feldolgozási technikák, például többszálúság vagy elosztott feladatsorok kihasználásával a rendszer optimalizálhatja az erőforrás-kihasználtságot és csökkentheti a teljes munkafolyamat végrehajtási idejét.

AI-alapú komponensek teljesítményének optimalizálása

Az AI-alapú komponensek, mint például a gépi tanulási modellek vagy a természetes nyelvfeldolgozó motorok, számításigényesek lehetnek és befolyásolhatják a munkafolyamat-vezérlő rendszer általános teljesítményét. Az AI komponensek teljesítményének optimalizálásához vegye figyelembe a következő technikákat:

1. **Gyorsítótárazás**: Ha az AI-feldolgozás tisztán generatív, és nem igényel valós idejű információlekérést vagy külső integrációkat a chat-válaszok generálásához, akkor érdemes megvizsgálni a gyorsítótárazási mechanizmusokat a gyakran használt vagy számításigényes műveletek eredményeinek tárolására és újrafelhasználására.

2. **Modell optimalizálás**: Folyamatosan optimalizálja az AI modellek használatát a munkafolyamat komponensekben. Ez magában foglalhatja olyan technikák alkalmazását, mint a *Prompt Desztilláció*, vagy egyszerűen az új modellek tesztelését, ahogy azok elérhetővé válnak.

3. **Kötegelt feldolgozás**: Ha GPT-4 osztályú modellekkel dolgozik, lehetősége lehet kötegelt feldolgozási technikák alkalmazására, hogy több adatpontot vagy

kérést egyetlen kötegben dolgozzon fel, ahelyett, hogy egyenként kezelné őket. Az adatok kötegekben történő feldolgozásával a rendszer optimalizálhatja az erőforrás-kihasználtságot és csökkentheti az ismétlődő modellkérések többletterhelését.

Teljesítmény monitorozása és profilozása

Az intelligens munkafolyamat-vezérlő rendszer teljesítményszűk keresztmetszeteinek azonosításához és a skálázhatóság optimalizálásához elengedhetetlen a monitorozási és profilozási mechanizmusok implementálása. Vegye figyelembe a következő megközelítéseket:

1. **Teljesítménymutatók**: Határozzon meg és kövessen nyomon kulcsfontosságú teljesítménymutatókat, mint például a válaszidő, áteresztőképesség, erőforrás-kihasználtság és késleltetés. Ezek a mutatók betekintést nyújtanak a rendszer teljesítményébe és segítenek azonosítani az optimalizálandó területeket. A népszerű AI modell aggregátor OpenRouter Host[1] és Speed[2] metrikákat tartalmaz minden API válaszban, így egyszerűvé téve ezeknek a kulcsfontosságú metrikáknak a követését.

2. **Profilozó eszközök**: Használjon profilozó eszközöket az egyes munkafolyamat-komponensek és AI műveletek teljesítményének elemzésére. A profilozó eszközök segíthetnek azonosítani a teljesítmény szűk keresztmetszeteit, nem hatékony kódrészleteket vagy erőforrás-igényes műveleteket. Népszerű profilozó eszközök közé tartozik a New Relic, Scout, vagy a programozási nyelv vagy keretrendszer által biztosított beépített profilozók.

3. **Terheléstesztelés**: Végezzen terheléstesztelést a rendszer teljesítményének értékeléséhez különböző szintű párhuzamos munkaterhelések mellett. A terheléstesztelés segít azonosítani a rendszer skálázhatósági korlátait, észlelni a teljesítményromlást, és biztosítani, hogy a rendszer képes kezelni a várható forgalmat a teljesítmény kompromittálása nélkül.

[1] A Host az az időtartam, ami az első bájt fogadásáig telt el a modell kiszolgálójától a streamelt generálás során, más néven "time to first byte."

[2] A Speed a befejező tokenek számának és a teljes generálási időnek a hányadosaként számítódik. Nem streamelt kérések esetén a késleltetés a generálási idő részének számít.

4. **Folyamatos monitorozás**: Implementáljon folyamatos monitorozási és riasztási mechanizmusokat a teljesítményproblémák és szűk keresztmetszetek proaktív észleléséhez. Állítson fel monitorozási irányítópultokat és riasztásokat a kulcsfontosságú teljesítménymutatók (KPI-k) követéséhez, és kapjon értesítéseket, amikor az előre meghatározott küszöbértékek átlépésre kerülnek. Ez lehetővé teszi a teljesítményproblémák gyors azonosítását és megoldását.

Skálázási stratégiák

Az intelligens munkafolyamat-vezérlő rendszer növekvő munkaterhelésének kezeléséhez és skálázhatóságának biztosításához vegye figyelembe a következő skálázási stratégiákat:

1. **Vertikális skálázás**: A vertikális skálázás az egyes csomópontok vagy szerverek erőforrásainak (pl. CPU, memória) növelését jelenti a nagyobb munkaterhelés kezeléséhez. Ez a megközelítés akkor alkalmas, amikor a rendszernek több feldolgozási teljesítményre vagy memóriára van szüksége a komplex munkafolyamatok vagy AI műveletek kezeléséhez.

2. **Horizontális skálázás**: A horizontális skálázás több csomópont vagy szerver hozzáadását jelenti a rendszerhez a munkaterhelés elosztása érdekében. Ez a megközelítés akkor hatékony, amikor a rendszernek nagy számú párhuzamos munkafolyamatot kell kezelnie, vagy amikor a munkaterhelés könnyen eloszlatható több csomópont között. A horizontális skálázás elosztott architektúrát és terheléselosztási mechanizmusokat igényel a forgalom egyenletes elosztásának biztosításához.

3. **Automatikus skálázás**: Implementáljon automatikus skálázási mechanizmusokat a csomópontok vagy erőforrások számának automatikus beállításához a munkaterhelés igénye alapján. Az automatikus skálázás lehetővé teszi, hogy a rendszer dinamikusan növelje vagy csökkentse a kapacitását a bejövő forgalom függvényében, biztosítva az optimális erőforrás-kihasználtságot és költséghatékonyságot. Az olyan felhőplatformok, mint az Amazon Web Services (AWS) vagy a Google Cloud Platform (GCP) olyan

automatikus skálázási képességeket biztosítanak, amelyeket ki lehet használni az intelligens munkafolyamat-vezérlő rendszerekhez.

Teljesítményoptimalizálási technikák

A skálázási stratégiák mellett vegye figyelembe a következő teljesítményoptimalizálási technikákat az intelligens munkafolyamat-vezérlő rendszer hatékonyságának növeléséhez:

1. **Hatékony adattárolás és -lekérés**: Optimalizálja a munkafolyamat-komponensek által használt adattárolási és -lekérési mechanizmusokat. Használjon hatékony adatbázis-indexelést, lekérdezés-optimalizálási technikákat és adat-gyorsítótárazást az adatintenzív műveletek késleltetésének minimalizálásához és a teljesítmény javításához.

2. **Aszinkron I/O**: Használjon aszinkron I/O műveleteket a blokkolás megakadályozására és a rendszer válaszképességének javítására. Az aszinkron I/O lehetővé teszi, hogy a rendszer párhuzamosan kezeljen több kérést anélkül, hogy várnia kellene az I/O műveletek befejezésére, ezáltal maximalizálva az erőforrások kihasználtságát.

3. **Hatékony szerializáció és deszerializáció**: Optimalizálja a munkafolyamat-komponensek közötti adatcseréhez használt szerializációs és deszerializációs folyamatokat. Használjon hatékony szerializációs formátumokat, mint például a Protocol Buffers vagy MessagePack, hogy csökkentse az adatszerializáció többletterhelését és javítsa a komponensek közötti kommunikáció teljesítményét.

Ruby-alapú alkalmazások esetén érdemes megfontolni a Universal ID használatát. A Universal ID a MessagePack és a Brotli kombinációját használja (egy sebességre és kiváló adattömörítésre optimalizált párosítás). Együttesen ezek a könyvtárak akár 30%-kal gyorsabbak, és a tömörítési arányuk csak 2-5%-kal marad el a Protocol Buffers-étől.

4. **Tömörítés és kódolás**: Alkalmazzon tömörítési és kódolási technikákat a munkafolyamat-komponensek között átvitt adatok méretének csökkentésére. A tömörítő algoritmusok, mint például a gzip vagy a Brotli, jelentősen csökkenthetik a hálózati sávszélesség használatát és javíthatják a rendszer általános teljesítményét.

Az intelligens munkafolyamat-vezérlő rendszerek tervezése és megvalósítása során a skálázhatósági és teljesítményszempontok figyelembevételével biztosítható, hogy a rendszer kezelni tudja a párhuzamos munkafolyamatok nagy számát, optimalizálja a mesterséges intelligencia által támogatott komponensek teljesítményét, és zökkenőmentesen skálázódjon a növekvő igények kielégítésére. A folyamatos monitorozás, profilozás és optimalizálás elengedhetetlen a rendszer teljesítményének és válaszképességének fenntartásához, ahogy a munkaterhelés és a komplexitás idővel növekszik.

Munkafolyamatok tesztelése és validálása

A tesztelés és validálás kritikus szempontjai az intelligens munkafolyamat-vezérlő rendszerek fejlesztésének és karbantartásának. A mesterséges intelligencia által támogatott munkafolyamatok összetett jellege miatt elengedhetetlen annak biztosítása, hogy minden komponens az elvárásoknak megfelelően működjön, a teljes munkafolyamat helyesen viselkedjen, és a mesterséges intelligencia döntései pontosak és megbízhatóak legyenek. Ebben a részben különböző technikákat és szempontokat vizsgálunk meg az intelligens munkafolyamatok teszteléséhez és validálásához.

Munkafolyamat-komponensek egységtesztelése

Az egységtesztelés során az egyes munkafolyamat-komponenseket elkülönítve teszteljük, hogy ellenőrizzük helyességüket és robusztusságukat. A mesterséges

intelligencia által támogatott munkafolyamat-komponensek egységtesztelése során vegye figyelembe a következőket:

1. **Bemenet validálása**: Tesztelje a komponens képességét különböző típusú bemenetek kezelésére, beleértve az érvényes és érvénytelen adatokat. Ellenőrizze, hogy a komponens megfelelően kezeli a szélsőséges eseteket és megfelelő hibaüzeneteket vagy kivételeket ad.

2. **Kimenet ellenőrzése**: Győződjön meg arról, hogy a komponens a várt kimenetet állítja elő egy adott bemeneti készlet esetén. Hasonlítsa össze a tényleges kimenetet az elvárt eredményekkel a helyesség biztosítása érdekében.

3. **Hibakezelés**: Tesztelje a komponens hibakezelési mechanizmusait különböző hibahelyzetek szimulálásával, mint például érvénytelen bemenet, erőforrás-elérhetetlenség vagy váratlan kivételek. Ellenőrizze, hogy a komponens megfelelően észleli és kezeli a hibákat.

4. **Határfeltételek**: Tesztelje a komponens viselkedését határfeltételek mellett, például üres bemenet, maximális bemeneti méret vagy szélsőséges értékek esetén. Győződjön meg arról, hogy a komponens megfelelően kezeli ezeket a feltételeket összeomlás vagy helytelen eredmények előállítása nélkül.

Íme egy példa egy munkafolyamat-komponens egységtesztjére Ruby-ban az RSpec tesztelési keretrendszer használatával:

```
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
```

Ebben a példában az `OrderValidator` komponenst két tesztesettel teszteljük: egy érvényes és egy érvénytelen rendelés esetével. A tesztesetek ellenőrzik, hogy a `validate` metódus a rendelés érvényességének megfelelően a várt logikai értéket adja-e vissza.

Munkafolyamatok közötti kapcsolatok integrációs tesztelése

Az integrációs tesztelés a különböző munkafolyamat-komponensek közötti kapcsolatok és adatáramlás ellenőrzésére összpontosít. Biztosítja, hogy a komponensek zökkenőmentesen működjenek együtt és a várt eredményeket produkálják. Az intelligens munkafolyamatok integrációs tesztelésénél vegye figyelembe a következőket:

1. **Komponensek közötti kapcsolat**: Tesztelje a munkafolyamat-komponensek közötti kommunikációt és adatcserét. Ellenőrizze, hogy az egyik komponens kimenete megfelelően kerül-e át a munkafolyamat következő komponensének bemenetére.

2. **Adatkonzisztencia**: Biztosítsa, hogy az adatok konzisztensek és pontosak maradjanak, miközben áthaladnak a munkafolyamaton. Ellenőrizze, hogy az adatátalakítások, számítások és összesítések helyesen kerülnek-e végrehajtásra.

3. **Kivételek továbbítása**: Tesztelje, hogyan történik a kivételek és hibák továbbítása és kezelése a munkafolyamat-komponensek között. Ellenőrizze, hogy a kivételek megfelelően el vannak-e kapva, naplózva vannak-e, és megfelelően kezelve vannak-e a munkafolyamat megszakításának elkerülése érdekében.

4. **Aszinkron viselkedés**: Ha a munkafolyamat aszinkron komponenseket vagy párhuzamos végrehajtást tartalmaz, tesztelje a koordinációs és szinkronizációs mechanizmusokat. Győződjön meg arról, hogy a munkafolyamat helyesen működik párhuzamos és aszinkron forgatókönyvek esetén is.

Íme egy példa egy munkafolyamat integrációs tesztjére Ruby-ban az RSpec tesztelési keretrendszer használatával:

```
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
```

Ebben a példában az `OrderProcessingWorkflow` tesztelése a különböző munkafolyamat-komponensek közötti kölcsönhatások ellenőrzésével történik. A

teszteset beállítja az egyes komponensek várható viselkedését, és biztosítja, hogy a munkafolyamat sikeresen feldolgozza a rendelést, ennek megfelelően frissítve a rendelés státuszát.

MI Döntési Pontok Tesztelése

Az MI döntési pontok tesztelése kulcsfontosságú az MI-alapú munkafolyamatok pontosságának és megbízhatóságának biztosításához. Az MI döntési pontok tesztelésekor vegye figyelembe a következőket:

1. **Döntési Pontosság**: Ellenőrizze, hogy az MI komponens pontos döntéseket hoz-e a bemeneti adatok és a betanított modell alapján. Hasonlítsa össze az MI döntéseit a várt eredményekkel vagy a referencia adatokkal.

2. **Szélsőséges Esetek**: Tesztelje az MI komponens viselkedését szélsőséges esetekben és szokatlan helyzetekben. Ellenőrizze, hogy az MI komponens megfelelően kezeli-e ezeket az eseteket, és ésszerű döntéseket hoz-e.

3. **Torzítás és Méltányosság**: Értékelje az MI komponenst a lehetséges torzítások szempontjából, és győződjön meg arról, hogy méltányos és elfogulatlan döntéseket hoz. Tesztelje a komponenst különböző bemeneti adatokkal, és elemezze az eredményeket diszkriminatív minták után kutatva.

4. **Magyarázhatóság**: Ha az MI komponens magyarázatokat vagy indoklást ad a döntéseihez, ellenőrizze a magyarázatok helyességét és érthetőségét. Győződjön meg arról, hogy a magyarázatok összhangban vannak az alapvető döntéshozatali folyamattal.

Íme egy példa egy MI döntési pont tesztelésére Ruby-ban az RSpec tesztelési keretrendszer használatával:

```
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
```

Ebben a példában a `FraudDetector` MI komponenst két tesztesettel vizsgáljuk: az egyik egy csalárd tranzakciót, a másik egy jogszerű tranzakciót tesztel. A tesztesetek ellenőrzik, hogy a `detect_fraud` metódus a tranzakció jellemzői alapján a várt logikai értéket adja-e vissza.

Végpontok közötti tesztelés

A végpontok közötti tesztelés magában foglalja a teljes munkafolyamat tesztelését elejétől a végéig, valós környezeti forgatókönyveket és felhasználói interakciókat szimulálva. Ez biztosítja, hogy a munkafolyamat megfelelően viselkedjen és a kívánt eredményeket produkálja. Az intelligens munkafolyamatok végpontok közötti tesztelésénél vegye figyelembe a következőket:

1. **Felhasználói forgatókönyvek**: Azonosítsa a gyakori felhasználói forgatókönyveket és tesztelje a munkafolyamat viselkedését ezekben a helyzetekben. Ellenőrizze, hogy a munkafolyamat megfelelően kezeli a felhasználói bemeneteket, megfelelő döntéseket hoz és a várt kimeneteket produkálja.

2. **Adatellenőrzés**: Győződjön meg arról, hogy a munkafolyamat ellenőrzi és tisztítja a felhasználói bemeneteket az adatinkonzisztenciák és biztonsági sebezhetőségek megelőzése érdekében. Tesztelje a munkafolyamatot különböző típusú bemeneti adatokkal, beleértve az érvényes és érvénytelen adatokat is.

3. **Hibaelhárítás**: Tesztelje a munkafolyamat képességét a hibák és kivételek kezelésére. Szimuláljon hibahelyzeteket és ellenőrizze, hogy a munkafolyamat megfelelően kezeli-e ezeket, naplózza-e a hibákat és megfelelő helyreállítási műveleteket végez-e.

4. **Teljesítmény és skálázhatóság**: Értékelje a munkafolyamat teljesítményét és skálázhatóságát különböző terhelési körülmények között. Tesztelje a munkafolyamatot nagy mennyiségű párhuzamos kéréssel, és mérje a válaszidőket, az erőforrás-kihasználtságot és a rendszer általános stabilitását.

Itt egy példa egy munkafolyamat végpontok közötti tesztelésére Ruby nyelven, az RSpec tesztelési keretrendszer és a Capybara könyvtár használatával a felhasználói interakciók szimulálásához:

```
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
```

Ebben a példában a végpontok közötti teszt egy felhasználót szimulál, aki megrendelést ad le a webes felületen keresztül. Kitölti a szükséges űrlapmezőket, elküldi a megrendelést, és ellenőrzi, hogy a megrendelés feldolgozása sikeres-e, megjelenítve a megfelelő visszaigazoló üzenetet és frissítve a megrendelés státuszát az adatbázisban.

Folyamatos Integráció és Telepítés

Az intelligens munkafolyamatok megbízhatóságának és karbantarthatóságának biztosítása érdekében ajánlott a tesztelést és validálást beépíteni a folyamatos integráció és telepítés (CI/CD) folyamatába. Ez lehetővé teszi a munkafolyamat-változtatások automatizált tesztelését és validálását, mielőtt azok az éles környezetbe kerülnének. Vegye figyelembe a következő gyakorlatokat:

1. **Automatizált Tesztvégrehajtás**: Konfigurálja a CI/CD folyamatot úgy, hogy automatikusan futtassa a tesztkészletet, amikor változtatások történnek a munkafolyamat kódbázisában. Ez biztosítja, hogy minden regressziót vagy hibát már a fejlesztési folyamat korai szakaszában észleljenek.

2. **Tesztlefedettség Monitorozása**: Mérje és kövesse nyomon a munkafolyamat-komponensek és a mesterséges intelligencia döntési pontjainak tesztlefedettségét. Törekedjen magas tesztlefedettségre annak biztosítása érdekében, hogy a kritikus útvonalakat és forgatókönyveket alaposan teszteljék.

3. **Folyamatos Visszajelzés**: Integrálja a teszteredményeket és a kódminőségi metrikákat a fejlesztési munkafolyamatba. Biztosítson folyamatos visszajelzést a fejlesztőknek a tesztek állapotáról, a kód minőségéről és a CI/CD folyamat során észlelt problémákról.

4. **Előkészítő Környezetek**: Telepítse a munkafolyamatot olyan előkészítő környezetekbe, amelyek szorosan tükrözik az éles környezetet. Végezzen további tesztelést és validálást az előkészítő környezetben, hogy észlelje az infrastruktúrával, konfigurációval vagy adatintegrációval kapcsolatos problémákat.

5. **Visszaállítási Mechanizmusok**: Implementáljon visszaállítási mechanizmusokat telepítési hibák vagy az éles környezetben észlelt kritikus problémák esetére. Biztosítsa, hogy a munkafolyamat gyorsan visszaállítható legyen egy korábbi stabil verzióra a leállási idő és a felhasználókra gyakorolt hatás minimalizálása érdekében.

Az intelligens munkafolyamatok fejlesztési életciklusába beépített tesztelés és validálás révén a szervezetek biztosíthatják MI-alapú rendszereik megbízhatóságát, pontosságát és karbantarthatóságát. A rendszeres tesztelés és validálás segít a hibák észlelésében, a regressziók megelőzésében és a munkafolyamat viselkedésébe és eredményeibe vetett bizalom kiépítésében.

2. rész: A minták

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Prompttervezés

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Gondolatmenet (Chain of Thought)

Hogyan működik

Példák

Tartalomgenerálás

Strukturált Entitás Létrehozás

LLM Ágensek Iránymutatása

Előnyök és Megfontolások

Módváltás

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Hogyan Működik

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Mikor használjuk

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Példa

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Szerepkiosztás

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Hogyan működik

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Mikor használjuk

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Példák

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Prompt Object

Hogyan működik

Prompt Sablon

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Hogyan működik

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Előnyök és megfontolások

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Mikor használjuk:

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Példa

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Structured IO

Hogyan működik

Strukturált IO skálázása

Előnyök és Megfontolások

Promptláncolás

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Hogyan Működik

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Mikor Használjuk

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Példa: Az Olympia betanítási folyamata

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Prompt Átíró

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Hogyan működik

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Példa

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Response Fencing

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Hogyan működik

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Előnyök és Megfontolások

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Hibakezelés

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Lekérdezéselemző

Hogyan működik

Implementáció

Szófaji címkézés (POS Tagging) és névelem-felismerés (NER)

Szándékosztályozás

Kulcsszókinyerés

Előnyök

Lekérdezés-átfogalmazó

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Hogyan Működik

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Példa

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Előnyök

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Hasbeszélő

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Hogyan Működik

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Mikor Használjuk

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Példa

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Diszkrét komponensek

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Predicate

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Hogyan működik

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Mikor használjuk

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Példa

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

API Homlokzat

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Hogyan Működik

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Fő Előnyök

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Mikor Használjuk

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Példa

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Hitelesítés és Jogosultságkezelés

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Kérésfeldolgozás

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Válaszformázás

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Hibakezelés és Szélsőséges Esetek

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Skálázhatósági és Teljesítménnyel Kapcsolatos Megfontolások

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Összehasonlítás Más Tervezési Mintákkal

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Eredményértelmező

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Hogyan Működik

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Mikor Használjuk

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Példa

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Virtuális Gép

Hogyan Működik

Mikor Használjuk

Példa

A Varázslat Mögött

Specifikáció és Tesztelés

A Viselkedés Specifikálása

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Tesztesetek Írása

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Példa: A Fordító Komponens Tesztelése

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

HTTP-interakciók visszajátszása

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Ember a Rendszerben (HITL)

Magas Szintű Minták

Hibrid Intelligencia

Adaptív Válasz

Ember-AI szerepváltás

Eszkaláció

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Hogyan működik

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Főbb előnyök

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Valós Alkalmazás: Egészségügy

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Visszacsatolási Hurok

Hogyan Működik

Alkalmazások és Példák

Fejlett Technikák az Emberi Visszajelzések Integrálásában

Passzív Információsugárzás

Hogyan Működik

Kontextuális Információmegjelenítés

Proaktív Értesítések

Magyarázó Betekintések

Interaktív Feltárás

Főbb Előnyök

Alkalmazások és Példák

Együttműködő Döntéshozatal (CDM)

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Hogyan Működik

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Példa

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Folyamatos Tanulás

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Hogyan működik

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Alkalmazások és példák

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Példa

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Etikai megfontolások

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

A HITL szerepe az AI kockázatok csökkentésében

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Technológiai fejlesztések és jövőkép

A HITL-rendszerek kihívásai és korlátai

Intelligens hibakezelés

Hagyományos hibakezelési megközelítések

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Kontextuális Hibaelemzés

Hogyan Működik

Prompttervezés a Kontextuális Hibaelemzéshez

Visszakereséssel bővített generálás a kontextuális hibadiagnosztikához

Intelligens hibajelentés

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Prediktív Hibamegelőzés

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Hogyan működik

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Intelligens Hibahelyreállítás

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Hogyan működik

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Személyre Szabott Hibakommunikáció

Hogyan Működik

Adaptív Hibakezelési Munkafolyamat

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Működési elv

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Minőségellenőrzés

Eval

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Probléma

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Megoldás

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Hogyan működik

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Példa

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Megfontolások

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Az aranystandardú referenciák megértése

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Hogyan működnek a referenciamentes kiértékelések

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Védőkorlát

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Probléma

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Megoldás

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Hogyan működik

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Példa

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Megfontolások

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Védőkorlátok és kiértékelések: Ugyanannak az érmének két oldala

A védőkorlátok és a referenciamentes kiértékelések felcserélhetősége

Kettős célú védőkorlátok és kiértékelések implementálása

Szójegyzék

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Szójegyzék

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

A

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

B

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

C

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

D

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

E

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

F

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

G

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

H

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

I

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

J

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

K

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

L

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

M

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

N

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

O

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

P

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Q

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

R

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

S

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

T

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

U

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

V

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

W

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Z

Ez a tartalom nem érhető el a minta részeként. A könyv megvásárolható a Leanpubon, a http://leanpub.com/patterns-of-application-development-using-ai-hu címen.

Index