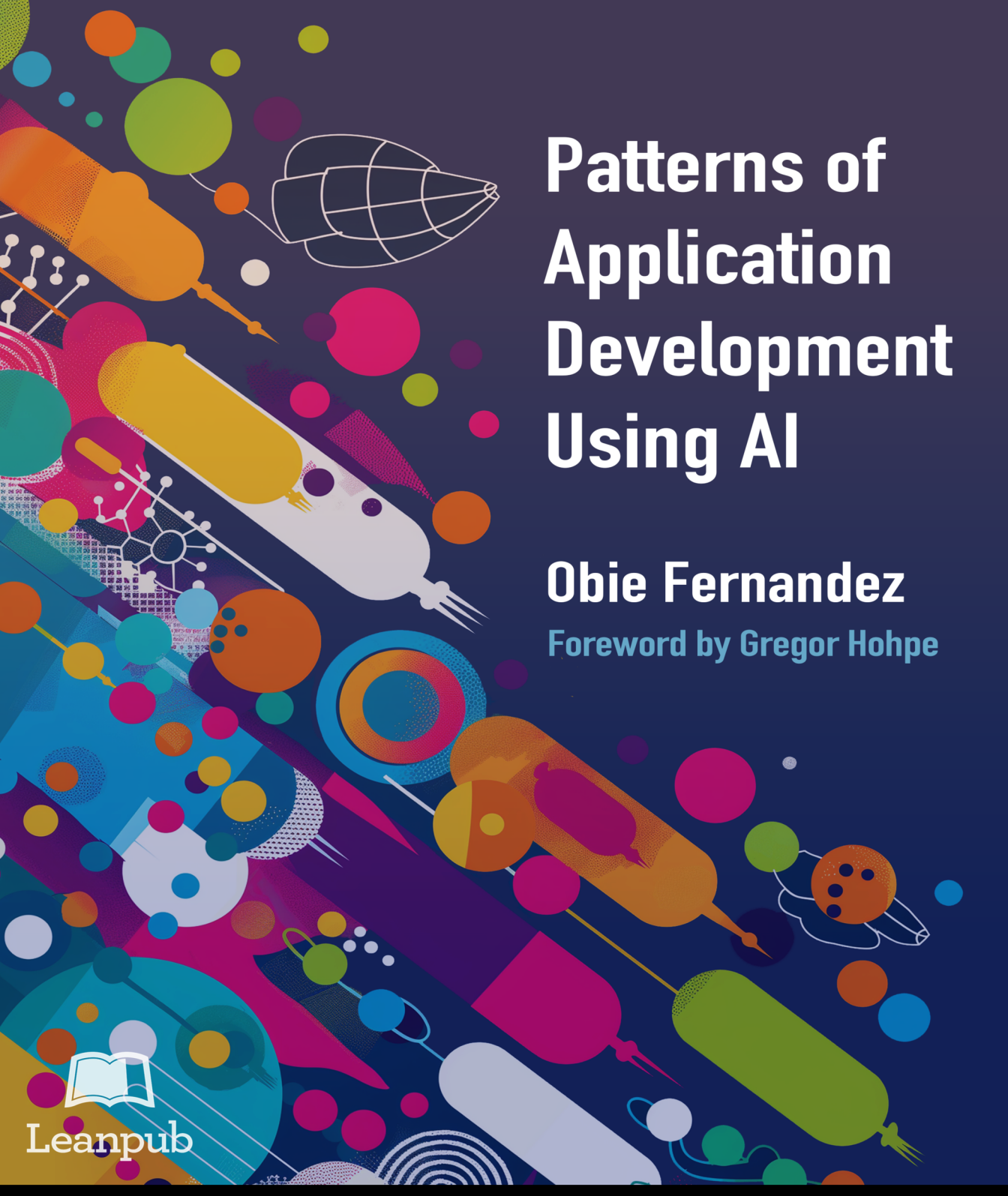


Українське Видання

Патерни Розробки Застосунків із Використанням ШІ (Українське Видання)

Obie Fernandez

Ця книга продається на
http://leanpub.com/patterns-of-application-development-using-ai-uk

Ця версія була опублікована 2025-01-23

Це книга Leanpub. Leanpub наділяє авторів та видавців можливостями здійснення процесу Lean Publishing. Lean Publishing — це процес публікації електронної книги, що ще формується, за допомогою простих у використанні інструментів та численних ітерацій для залучення відгуків читачів, коригування курсу до створення ідеальної книги і здобування розголосу після її завершення.

Твітніть про цю книгу!

Будь ласка, допоможіть Obie Fernandez розповсюдити інформацію про цю книгу на Twitter!

Рекомендований хештег для цієї книги #poaduai.

Дізнайтеся, що інші люди кажуть про книгу, натиснувши на це посилання, щоб шукати цей хештег на Twitter:

#poaduai

Моїй незламній королеві, моїй музі, моєму світлу та коханню, Вікторії

Also By 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

Зміст

Передмова від Грегора Хопе

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Передмова

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Про книгу

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Про приклади коду

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Що я не охоплюю

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Для кого ця книга

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Побудова спільного словника

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Долучайтесь

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Подяки

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Що з ілюстраціями?

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

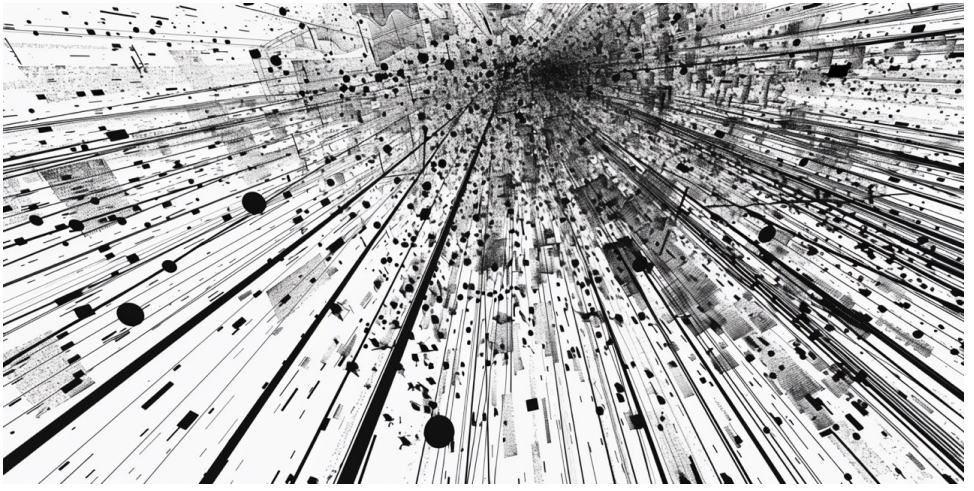
Про Lean Publishing

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Про автора

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Вступ



Якщо ви прагнете почати інтегрувати Великі Мовні Моделі (ВММ) у свої програмні проєкти, сміливо переходьте до патернів та прикладів коду, представлених у наступних розділах. Однак, щоб повністю оцінити силу та потенціал цих патернів, варто приділити час розумінню ширшого контексту та цілісного підходу, який вони представляють.

Ці патерни — це не просто набір ізольованих технік, а єдина структура для інтеграції ШІ у ваші застосунки. Я використовую Ruby on Rails, але ці патерни повинні працювати практично в будь-якому іншому програмному середовищі. Вони охоплюють широкий спектр питань, від керування даними та оптимізації продуктивності до користувацького досвіду та безпеки, надаючи комплексний набір інструментів для вдосконалення традиційних практик програмування можливостями ШІ.

Кожна категорія патернів вирішує конкретну проблему або можливість, що виникає під час інтеграції компонентів ШІ у ваш застосунок. Розуміючи взаємозв'язки та синергію між цими патернами, ви зможете приймати обґрунтовані рішення щодо того, де і як найефективніше застосовувати ШІ.

Патерни ніколи не є прескриптивними рішеннями і не повинні розглядатися як такі. Вони задумані як адаптивні будівельні блоки, які слід налаштовувати відповідно до унікальних вимог та обмежень вашого власного унікального застосунку. Успішне застосування цих патернів (як і будь-яких інших у сфері програмного забезпечення) залежить від глибокого розуміння предметної області, потреб користувачів та загальної технічної архітектури вашого проєкту.

Думки про архітектуру програмного забезпечення

Я почав програмувати у 1980-х роках і був залучений до хакерської спільноти, і ніколи не втрачав хакерського мислення, навіть ставши професійним розробником програмного забезпечення. З самого початку я завжди мав здоровий скептицизм щодо того, яку цінність насправді приносили архітектори програмного забезпечення у своїх вежах зі слонової кістки.

Одна з причин, чому я особисто так захоплений змінами, які принесла ця потужна нова хвиля технологій ШІ, — це її вплив на те, що ми вважаємо рішеннями в *архітектурі програмного забезпечення*. Вона кидає виклик традиційним уявленням про те, що є "правильним" способом проєктування та реалізації наших програмних проєктів. Вона також ставить під сумнів, чи можна все ще розглядати архітектуру переважно як *ті частини системи, які важко змінити*, оскільки вдосконалення ШІ робить як ніколи простим внесення змін до будь-якої частини вашого проєкту в будь-який час.

Можливо, ми входимо в пікові роки “постмодерного” підходу до розробки програмного забезпечення. У цьому контексті постмодерний підхід означає фундаментальний відхід від традиційних парадигм, де розробники відповідали за написання та підтримку кожного рядка коду. Натомість він охоплює ідею делегування завдань, таких як маніпуляції з даними, складні алгоритми і навіть цілі частини логіки застосунку, стороннім бібліотекам та зовнішнім API. Цей постмодерний зсув представляє значний відхід від загальноприйнятої мудрості побудови застосунків з нуля і спонукає розробників переосмислити свою роль у процесі розробки.

Я завжди вірив, що хороші програмісти пишуть лише той код, який абсолютно необхідно написати, базуючись на вченнях Larry Wall та інших хакерських світил, подібних до нього. Мінімізуючи кількість написаного коду, ми можемо рухатися швидше, зменшити поверхню для помилок, спростити обслуговування та покращити загальну надійність наших застосунків. Менша кількість коду дозволяє нам зосередитися на основній бізнес-логіці та досвіді користувачів, делегуючи іншу роботу іншим сервісам.

Тепер, коли системи на базі ШІ можуть виконувати завдання, які раніше були виключною прерогативою коду, написаного людиною, ми повинні бути ще продуктивнішими та гнучкішими, з ще більшим фокусом на створенні бізнес-цінності та користувацького досвіду.

Звісно, делегування величезних частин вашого проєкту системам ШІ має свої компроміси, такі як потенційна втрата контролю та необхідність надійних механізмів моніторингу та зворотного зв’язку. Саме тому це вимагає нового набору навичок та знань, включаючи принаймні деяке фундаментальне розуміння того, як працює ШІ.

Що таке Велика Мовна Модель?

Великі Мовні Моделі (ВММ) — це тип штучного інтелекту, який привернув значну увагу в останні роки, особливо після запуску GPT-3 компанією OpenAI у 2020 році. ВММ розроблені для обробки, розуміння та генерації людської мови з винятковою точністю та плавністю. У цьому розділі ми коротко розглянемо, як працюють ВММ і чому вони добре підходять для побудови інтелектуальних системних компонентів.

В своїй основі ВММ базуються на алгоритмах глибинного навчання, зокрема нейронних мережах. Ці мережі складаються з взаємопов'язаних вузлів, або нейронів, які обробляють та передають інформацію. Архітектурою вибору для ВММ часто є модель трансформер, яка довела свою високу ефективність в обробці послідовних даних, таких як текст.

Трансформерні моделі базуються на механізмі уваги і переважно використовуються для завдань, пов'язаних з послідовними даними, наприклад, для обробки природної мови. Трансформери обробляють вхідні дані всі одразу, а не послідовно, що дозволяє їм ефективніше охоплювати залежності на великих відстанях. Вони мають шари механізмів уваги, які допомагають моделі фокусуватися на різних частинах вхідних даних для розуміння контексту та взаємозв'язків.

Процес навчання великих мовних моделей включає знайомство моделі з величезними обсягами текстових даних, таких як книги, статті, веб-сайти та репозиторії коду. Під час навчання модель вчиться розпізнавати шаблони, взаємозв'язки та структури в тексті. Вона охоплює статистичні властивості мови, такі як граматичні правила, словесні асоціації та контекстуальні значення.

Однією з ключових технік, що використовуються у навчанні великих мовних моделей, є навчання без нагляду. Це означає, що модель навчається з даних без явної розмітки чи керування. Вона самостійно виявляє шаблони та

представлення, аналізуючи спільну появу слів і фраз у навчальних даних. Це дозволяє великим мовним моделям розвинути глибоке розуміння мови та її тонкощів.

Іншим важливим аспектом великих мовних моделей є їхня здатність працювати з *контекстом*. При обробці тексту великі мовні моделі враховують не лише окремі слова, а й навколишній контекст. Вони беруть до уваги попередні слова, речення і навіть абзаци для розуміння значення та наміру тексту. Це контекстуальне розуміння дозволяє великим мовним моделям генерувати узгоджені та релевантні відповіді. Одним з основних способів оцінки можливостей конкретної моделі великої мовної моделі є розгляд розміру контексту, який вони можуть враховувати для генерації відповідей.

Після навчання великі мовні моделі можна використовувати для широкого спектру мовних завдань. Вони можуть генерувати текст, схожий на людський, відповідати на запитання, узагальнювати документи, перекладати мови і навіть писати код. Універсальність великих мовних моделей робить їх цінними для створення компонентів інтелектуальних систем, які можуть взаємодіяти з користувачами, обробляти та аналізувати текстові дані та генерувати змістовні результати.

Включаючи великі мовні моделі в архітектуру додатків, ви можете створювати компоненти штучного інтелекту, які розуміють і обробляють користувацький ввід, генерують динамічний контент та надають інтелектуальні рекомендації чи дії. Але робота з великими мовними моделями вимагає ретельного врахування вимог до ресурсів та компромісів продуктивності. Великі мовні моделі вимагають значних обчислювальних потужностей та пам'яті (іншими словами, грошей) для роботи. Більшості з нас доведеться оцінити фінансові наслідки інтеграції великих мовних моделей у наші додатки та діяти відповідно.

Розуміння виведення

Виведення стосується процесу, за допомогою якого модель генерує передбачення або результати на основі нових, невідомих даних. Це фаза, коли навчена модель використовується для прийняття рішень або генерації тексту, зображень чи іншого контенту у відповідь на користувацький ввід.

Під час фази навчання модель штучного інтелекту вчиться на великому наборі даних, налаштовуючи свої параметри для мінімізації помилок у своїх передбаченнях. Після навчання модель може застосовувати вивчене до нових даних. Виведення - це спосіб, яким модель використовує свої вивчені шаблони та знання для генерації результатів.

Для великих мовних моделей виведення включає отримання запиту чи вхідного тексту та створення узгодженої та контекстуально релевантної відповіді у вигляді потоку *токенів* (про які ми поговоримо незабаром). Це може бути відповідь на запитання, завершення речення, генерація історії чи переклад тексту, серед багатьох інших завдань.

На відміну від того, як думаємо ви і я, "мислення" моделі штучного інтелекту через виведення відбувається в одній безстановій операції. Тобто її мислення обмежене процесом генерації. Вона буквально має думати вголос, ніби я поставив вам запитання і приймав відповідь від вас лише у стилі "потоку свідомості".

Великі мовні моделі бувають різних розмірів і різновидів

Хоча практично всі популярні великі мовні моделі базуються на одній і тій же базовій архітектурі трансформера і навчаються на величезних текстових наборах даних, вони бувають різних розмірів і точно налаштовуються для різних

цілей. Розмір великої мовної моделі, що вимірюється кількістю параметрів у її нейронній мережі, має великий вплив на її можливості. Більші моделі з більшою кількістю параметрів, як GPT-4, який, за чутками, має від 1 до 2 трильйонів параметрів, зазвичай більш обізнані та здібні, ніж менші моделі. Однак більші моделі також потребують набагато більше обчислювальної потужності для роботи, що призводить до більших витрат при їх використанні через API-виклики.

Щоб зробити великі мовні моделі більш практичними та пристосованими для конкретних випадків використання, базові моделі часто точно налаштовуються на більш цільових наборах даних. Наприклад, велика мовна модель може бути навчена на великому корпусі діалогів для спеціалізації в розмовному штучному інтелекті. Інші навчаються на коді, щоб наділити їх знаннями програмування. Існують навіть моделі, які спеціально навчені для рольових взаємодій з користувачами!

Моделі на основі пошуку vs Генеративні моделі

У світі великих мовних моделей (ВММ) існує два основних підходи до генерації відповідей: моделі на основі пошуку та генеративні моделі. Кожен підхід має свої сильні та слабкі сторони, і розуміння відмінностей між ними може допомогти вам вибрати правильну модель для вашого конкретного випадку використання.

Моделі на основі пошуку

Моделі на основі пошуку, також відомі як моделі пошуку інформації, генерують відповіді шляхом пошуку у великій базі даних наявного тексту та вибору найбільш релевантних уривків на основі вхідного запиту. Ці моделі не генерують новий текст з нуля, а натомість зшивають разом уривки з бази даних для формування зв’язної відповіді.

Однією з головних переваг моделей на основі пошуку є їхня здатність надавати фактично точну та актуальну інформацію. Оскільки вони спираються на базу даних курованого тексту, вони можуть витягувати релевантну інформацію з надійних джерел і представляти її користувачеві. Це робить їх добре придатними для застосунків, які вимагають точних, фактичних відповідей, таких як системи запитань-відповідей або бази знань.

Проте моделі на основі пошуку мають певні обмеження. Вони настільки хороші, наскільки хороша база даних, яку вони шукають, тому якість і охоплення бази даних безпосередньо впливають на продуктивність моделі. Крім того, ці моделі можуть мати труднощі з генерацією зв'язних і природньо звучних відповідей, оскільки вони обмежені текстом, доступним у базі даних.

У цій книзі ми не розглядаємо використання чистих моделей пошуку.

Генеративні моделі

Генеративні моделі, з іншого боку, створюють новий текст з нуля на основі шаблонів і зв'язків, які вони вивчили під час навчання. Ці моделі використовують своє розуміння мови для генерації нових відповідей, які адаптовані до вхідного запиту.

Головною перевагою генеративних моделей є їхня здатність створювати креативний, зв'язний і контекстуально релевантний текст. Вони можуть вести відкриті розмови, генерувати історії і навіть писати код. Це робить їх ідеальними для застосунків, які вимагають більш відкритих і динамічних взаємодій, таких як чат-боти, створення контенту та помічники з креативного письма.

Однак генеративні моделі іноді можуть створювати суперечливу або фактично неправильну інформацію, оскільки вони спираються на шаблони, вивчені під час навчання, а не на куровану базу даних фактів. Вони також можуть бути більш схильними до упереджень і галюцинацій, генеруючи текст, який є правдоподібним, але не обов'язково істинним.

Прикладами генеративних ВММ є серія GPT від OpenAI (GPT-3, GPT-4) та Claude від Anthropic.

Гібридні моделі

Кілька комерційно доступних ВММ поєднують обидва підходи - пошуковий та генеративний - у гібридній моделі. Ці моделі використовують методи пошуку для знаходження релевантної інформації з бази даних, а потім використовують генеративні методи для синтезу цієї інформації у зв'язну відповідь.

Гібридні моделі прагнуть поєднати фактичну точність моделей на основі пошуку з можливостями генерації природної мови генеративних моделей. Вони можуть надавати більш надійну та актуальну інформацію, зберігаючи при цьому здатність вести відкриті розмови.

При виборі між моделями на основі пошуку та генеративними моделями слід враховувати конкретні вимоги вашого застосунку. Якщо основною метою є надання точної, фактичної інформації, модель на основі пошуку може бути найкращим вибором. Якщо застосунок вимагає більш відкритих і креативних взаємодій, генеративна модель може бути більш підходящою. Гібридні моделі пропонують баланс між двома підходами і можуть бути хорошим вибором для застосунків, які вимагають як фактичної точності, так і генерації природної мови.

Зрештою, вибір між моделями на основі пошуку та генеративними моделями залежить від конкретного випадку використання та компромісів між точністю, креативністю та гнучкістю. Розуміючи сильні сторони та обмеження кожного підходу, ви можете приймати обґрунтовані рішення.

Навчання на інструкціях

Навчання на інструкціях є підмножиною розмовного навчання, яке використовує написані людиною запити та відповіді, щоб покращити здатність ВММ розуміти та виконувати інструкції. Ось кілька простих прикладів, що ілюструють, як навчання на інструкціях на написаних людиною запитах та відповідях покращує здатність ВММ розуміти та виконувати інструкції:

Приклад 1: Підтримка клієнтів

Інструкція: “Надайте відповідь клієнту, який просить повернення коштів.”

Написаний людиною запит: “Я хотів би попросити повернення коштів за мою нещодавню покупку. Продукт не виправдав моїх очікувань.”

Написана людиною відповідь: “Мені шкода чути, що продукт не виправдав ваших очікувань. Щоб обробити повернення коштів, будь ласка, надайте номер вашого замовлення та причину повернення. Ми розглянемо ваш запит і повернемося до вас протягом 2-3 робочих днів.”

Приклад 2: Освітня допомога

Інструкція: “Поясніть концепцію фотосинтезу 10-річній дитині.”

Запит від людини: “Чи можеш пояснити, як рослини виробляють їжу?”

Відповідь від людини: “Звісно! Рослини виробляють їжу через процес, який називається фотосинтезом. Вони використовують сонячне світло, воду та вуглекислий газ для створення їжі та кисню. У листях рослин є особлива частина, яка називається хлорофіл, що допомагає їм поглинати сонячне

світло."

Приклад 3: Особистий асистент

Інструкція: "Заплануй зустріч на наступний вівторок о 15:00."

Запит від людини: "Чи можеш запланувати зустріч на вівторок о 15:00?"

Відповідь від людини: "Звісно, я запланував(ла) зустріч для вас на наступний вівторок о 15:00. Чи потрібно вам щось ще?"

Результатом є різноманітна екосистема ВММ (великих мовних моделей) різних розмірів та з різними спеціалізаціями. Менші моделі в діапазоні 1-7 мільярдів параметрів забезпечують хороші загальні мовні можливості, будучи при цьому ефективнішими в роботі.

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

Моделі середнього розміру близько 30-70 мільярдів параметрів пропонують потужніші можливості міркування та виконання інструкцій.

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

При виборі ВММ для інтеграції в додаток необхідно збалансувати можливості моделі з практичними факторами, такими як вартість, затримка, довжина контексту та фільтрація контенту. Менші моделі, налаштовані на інструкції, часто є найкращим вибором для простіших мовних завдань, тоді як найбільші

моделі можуть знадобитися для складного міркування чи аналізу. Навчальні дані моделі також є важливим фактором, оскільки вони визначають дату відсічення знань моделі.

Деякі моделі, як-от деякі від Perplexity, підключені до джерел інформації в реальному часі, тому вони фактично не мають дати відсічення. Коли ви ставите їм запитання, вони можуть самостійно вирішувати здійснювати веб-пошук і отримувати довільні веб-сторінки для генерації відповіді.

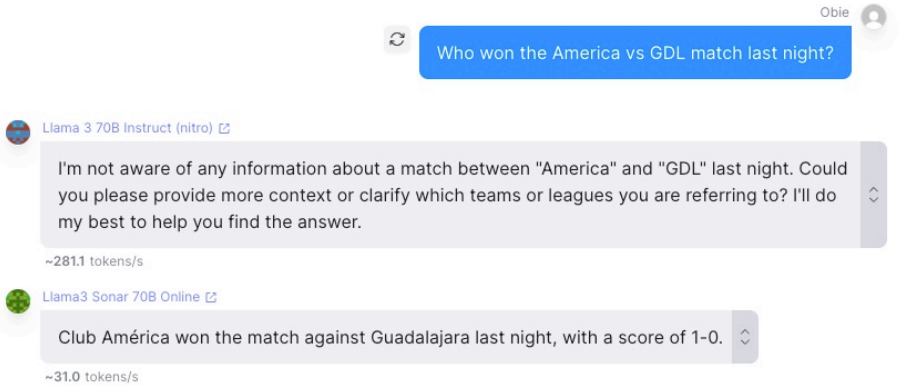


Рисунок 1. Llama3 з та без онлайн-доступу

Зрештою, не існує універсальної ВММ. Розуміння відмінностей у розмірі моделі, архітектурі та навчанні є ключовим для вибору правильної моделі для конкретного випадку використання. Експериментування з різними моделями - єдиний практичний спосіб виявити, які з них забезпечують найкращу продуктивність для поставленого завдання.

Токенізація: Розбиття тексту на частини

Перш ніж велика мовна модель зможе обробити текст, цей текст потрібно розбити на менші одиниці, які називаються *токенами*. Токени можуть бути окремими словами, частинами слів або навіть окремими символами. Процес розбиття тексту на токени називається токенізацією, і це важливий крок у підготовці даних для мовної моделі.

Рисунок 2. Це речення містить 27 токенів

Різні ВММ використовують різні стратегії токенізації, які можуть суттєво впливати на продуктивність та можливості моделі. Деякі поширені токенізатори, що використовуються ВММ, включають:

- **GPT (Побайтове парне кодування)**: Токенізатори GPT використовують метод, який називається побайтовим парним кодуванням (BPE) для розбиття тексту на підслівні одиниці. BPE послідовно об'єднує найчастіші пари байтів у текстовому корпусі, формуючи словник підслівних токенів. Це дозволяє токенізатору обробляти рідкісні та нові слова, розбиваючи їх на більш поширені підслівні частини. Токенізатори GPT використовуються такими моделями, як GPT-3 та GPT-4.

- **Llama (SentencePiece)**: Токенізатори Llama використовують бібліотеку SentencePiece, що є бібліотекою для неконтрольованої токенізації та детокенізації тексту. SentencePiece обробляє вхідний текст як послідовність символів Unicode і навчається словнику підслів на основі тренувального корпусу. Він може обробляти будь-яку мову, яку можна закодувати в Unicode, що робить його придатним для багатомовних моделей.

Токенізатори Llama використовуються такими моделями, як Llama та Alpaca від Meta.

- **SentencePiece (Unigram)**: Токенізатори SentencePiece також можуть використовувати інший алгоритм під назвою Unigram, який базується на техніці регуляризації підслів. Токенізація Unigram визначає оптимальний словник підслів на основі унограмної мовної моделі, яка призначає ймовірності окремим підсловам. Цей підхід може створювати більш семантично значущі підслова порівняно з BPE. SentencePiece з Unigram використовується такими моделями, як T5 та BERT від Google.

- **Google Gemini (Мультимодальна токенізація)**: Google Gemini використовує схему токенізації, розроблену для обробки різних типів даних, включаючи текст, зображення, аудіо, відео та код. Ця мультимодальна здатність дозволяє Gemini обробляти та інтегрувати різні форми інформації. Примітно, що Google Gemini 1.5 Pro має контекстне вікно, яке може обробляти мільйони токенів, що значно більше, ніж попередні моделі. Таке велике контекстне вікно дозволяє моделі обробляти більший контекст, потенційно призводячи до точніших відповідей. Однак важливо зазначити, що схема токенізації Gemini набагато ближча до одного токена на символ, ніж в інших моделях. Це означає, що фактична вартість використання моделей Gemini може бути значно вищою, ніж очікувалося, якщо ви звикли використовувати такі моделі, як GPT, оскільки ціноутворення Google базується на символах, а не на токенах.

Вибір токенізатора впливає на кілька аспектів LLM, включаючи:

- **Розмір словника**: Токенізатор визначає розмір словника моделі, тобто набір унікальних токенів, які вона розпізнає. Більший, більш деталізований словник може допомогти моделі обробляти ширший діапазон слів і фраз і

навіть стати мультимодальною (здатною розуміти та генерувати не лише текст), але це також збільшує вимоги до пам'яті моделі та обчислювальну складність.

- **Обробка рідкісних та невідомих слів**: Токенізатори, які використовують підслова, такі як BPE та SentencePiece, можуть розбивати рідкісні та невідомі слова на більш поширені частини підслів. Це дозволяє моделі робити обґрунтовані припущення про значення слів, які вона раніше не бачила, на основі підслів, з яких вони складаються.
- **Багатомовна підтримка**: Токенізатори на кшталт SentencePiece, які можуть обробляти будь-яку мову, що кодується в Unicode, добре підходять для багатомовних моделей, яким потрібно обробляти текст різними мовами.

При виборі LLM для конкретного застосування важливо враховувати токенізатор, який вона використовує, та наскільки добре він відповідає конкретним потребам обробки мови для поставленого завдання. Токенізатор може мати значний вплив на здатність моделі обробляти специфічну термінологію предметної області, рідкісні слова та багатомовний текст.

Розмір контексту: Скільки інформації може використовувати мовна модель під час виведення?

При обговоренні мовних моделей розмір контексту відноситься до обсягу тексту, який модель може враховувати при обробці або генерації своїх відповідей. По суті, це міра того, скільки інформації модель може "пам'ятати" і використовувати для формування своїх виводів (виражається в токенах). Розмір контексту мовної моделі може мати значний вплив на її можливості та типи завдань, які вона може ефективно виконувати.

Що таке розмір контексту?

З технічної точки зору, розмір контексту визначається кількістю токенів (слів або частин слів), які мовна модель може обробити в одній вхідній послідовності. Це часто називають “діапазоном уваги” або “контекстним вікном” моделі. Чим більший розмір контексту, тим більше тексту модель може одночасно враховувати при генерації відповіді або виконанні завдання.

Різні мовні моделі мають різні розміри контексту, від кількох сотень токенів до мільйонів токенів. Для довідки, типовий абзац тексту може містити близько 100-150 токенів, тоді як ціла книга може містити десятки або сотні тисяч токенів.

> Існують навіть роботи над ефективними методами масштабування великих мовних моделей (LLM) на основі трансформерів для нескінченно довгих входів з обмеженою пам’яттю та обчисленнями.

Чому розмір контексту важливий?

Розмір контексту мовної моделі має значний вплив на її здатність розуміти та генерувати зв’язний, контекстуально релевантний текст. Ось кілька ключових причин, чому розмір контексту має значення:

1. **Розуміння довгих текстів**: Моделі з більшим розміром контексту краще розуміють та аналізують довші тексти, такі як статті, звіти чи навіть цілі книги. Це критично важливо для таких завдань, як узагальнення документів, відповіді на запитання та аналіз контенту.

2. **Підтримка зв’язності**: Більше контекстне вікно дозволяє моделі підтримувати зв’язність та послідовність у довших фрагментах виводу.

Це важливо для таких завдань, як генерація історій, діалогові системи та створення контенту, де підтримка послідовної розповіді чи теми є необхідною. Це також абсолютно критично при використанні ВММ для генерації чи трансформації структурованих даних.

3. **Охоплення далекосяжних залежностей:** Деякі мовні завдання вимагають розуміння зв'язків між словами чи фразами, які знаходяться далеко одне від одного в тексті. Моделі з більшим розміром контексту краще здатні охоплювати ці далекосяжні залежності, що може бути важливим для таких завдань як аналіз настроїв, переклад та розуміння мови.
4. **Обробка складних інструкцій**: У застосуваннях, де мовні моделі використовуються для виконання складних, багатокрокових інструкцій, більший розмір контексту дозволяє моделі враховувати весь набір інструкцій при генерації відповіді, а не лише останні кілька слів.

Приклади мовних моделей з різними розмірами контексту

Ось кілька прикладів мовних моделей з різними розмірами контексту:

- OpenAI GPT-3.5 Turbo: 4,095 токенів
- Mistral 7B Instruct: 32,768 токенів
- Anthropic Claude v1: 100,000 токенів
- OpenAI GPT-4 Turbo: 128,000 токенів
- Anthropic Claude v2: 200,000 токенів
- Google Gemini Pro 1.5: 2.8M токенів

Як бачите, існує широкий діапазон розмірів контексту серед цих моделей, від приблизно 4,000 токенів для моделі OpenAI GPT-3.5 Turbo до 200,000 токенів для моделі Anthropic Claude v2. Деякі моделі, як-от Google PaLM 2 та OpenAI GPT-4, пропонують різні варіанти з більшими розмірами контексту (наприклад, версії

“32k”), які можуть обробляти ще довші вхідні послідовності. І на даний момент (квітень 2024) Google Gemini Pro може похвалитися майже 3 мільйонами токенів!

Варто зазначити, що розмір контексту може відрізнятися залежно від конкретної реалізації та версії певної моделі. Наприклад, оригінальна модель OpenAI GPT-4 має розмір контексту 8,191 токенів, тоді як пізніші варіанти GPT-4, такі як Turbo та 4o, мають значно більший розмір контексту - 128,000 токенів.

> Сем Альтман порівняв поточні обмеження контексту з кілобайтами оперативної пам’яті, з якими доводилося мати справу програмістам персональних комп’ютерів у 80-х роках, і сказав, що в найближчому майбутньому ми зможемо вмістити “всі ваші особисті дані” в контекст великої мовної моделі.

Вибір правильного розміру контексту

При виборі мовної моделі для конкретного застосування важливо враховувати вимоги до розміру контексту для поставленого завдання. Для завдань, що включають короткі, ізольовані фрагменти тексту, як-от аналіз настроїв чи прості відповіді на запитання, може бути достатньо меншого розміру контексту. Однак для завдань, що вимагають розуміння та генерації довших, складніших текстів, ймовірно, буде необхідний більший розмір контексту.

Варто зазначити, що більші розміри контексту часто пов’язані з підвищеними обчислювальними витратами та повільнішою обробкою, оскільки модель повинна враховувати більше інформації при генерації відповіді. Таким чином, при виборі мовної моделі для вашого застосування необхідно знайти баланс між розміром контексту та продуктивністю.

> Чому б просто не вибрати модель з найбільшим розміром контексту і не завантажити її максимальною кількістю інформації? Ну, крім факторів продуктивності, іншим головним міркуванням є вартість. У березні 2024 року один цикл запит-відповідь з використанням Google Gemini Pro 1.5 з повним контекстом коштуватиме вам майже 8 доларів США. Якщо у вас є варіант використання, який виправдовує ці витрати - чудово! Але для більшості застосувань це просто занадто дорого на порядки величини.

Пошук Голок у Стогах Сіна

Концепція пошуку голки в стозі сіна давно стала метафорою для викликів пошуку в великих наборах даних. У сфері ВММ ми трохи змінюємо цю аналогію. Уявіть, що ми шукаємо не просто один факт, захований у величезному тексті (як повна антологія есе Paul Graham), а кілька фактів, розкиданих по всьому тексту. Цей сценарій більше схожий на пошук кількох голок на величезному полі, а не просто в одному стозі сіна. І ось у чому справа: нам потрібно не лише знайти ці голки, але й зв'язати їх у логічну нитку.

Коли перед ВММ постає завдання пошуку та міркування про кілька фактів, вбудованих у довгі контексти, вони стикаються з подвійним викликом. По-перше, існує пряма проблема точності пошуку — вона природно знижується зі збільшенням кількості фактів. Це очікувано; зрештою, відстеження багатьох деталей у розлогому тексті випробовує навіть найскладніші моделі.

По-друге, і можливо більш критично, це виклик міркування з цими фактами. Одна справа — виокремити факти; інша — синтезувати їх у зв'язну розповідь чи відповідь. Ось де починається справжнє випробування. Ефективність ВММ у завданнях міркування має тенденцію до більшої деградації, ніж у простих завданнях пошуку. Ця деградація стосується не лише обсягу; це про складний

танець контексту, релевантності та умовиводів.

Чому це відбувається? Розглянемо динаміку пам'яті та уваги в людському пізнанні, яка певною мірою відображається у ВММ. При обробці великих обсягів інформації ВММ, як і люди, можуть втрачати попередні деталі, поглинаючи нові. Це особливо помітно в моделях, які не розроблені спеціально для автоматичного пріоритезування чи повернення до попередніх сегментів тексту.

Більше того, здатність ВММ поєднувати ці знайдені факти у зв'язну відповідь подібна до побудови наративу. Це вимагає не просто пошуку інформації, а глибокого розуміння та контекстуального розміщення, що залишається серйозним викликом для сучасного ШІ.

То що це означає для нас як розробників та інтеграторів цих технологій? Нам потрібно чітко усвідомлювати ці обмеження при проєктуванні систем, які покладаються на ВММ для виконання складних завдань з довгими текстами. Розуміння того, що продуктивність може знижуватися за певних умов, допомагає нам встановлювати реалістичні очікування та розробляти кращі резервні механізми чи додаткові стратегії.

Модальності: За Межами Тексту

Хоча більшість мовних моделей сьогодні зосереджені на обробці та генерації тексту, спостерігається зростаюча тенденція до мультимодальних моделей, які можуть природно приймати та виводити різні типи даних, такі як зображення, аудіо та відео. Ці мультимодальні моделі відкривають нові можливості для застосунків на базі ШІ, які можуть розуміти та генерувати контент у різних модальностях.

Що Таке Модальності?

У контексті мовних моделей модальності відносяться до різних типів даних, які модель може обробляти та генерувати. Найпоширенішою модальністю є текст,

який включає письмову мову в різних формах, як-от книги, статті, веб-сайти та дописи в соціальних мережах. Однак існує кілька інших модальностей, які все частіше включаються до мовних моделей:

- **Зображення**: Візуальні дані, такі як фотографії, ілюстрації та діаграми.
- **Аудіо**: Звукові дані, такі як мовлення, музика та навколишні звуки.
- **Відео**: Рухомі візуальні дані, часто супроводжувані аудіо, такі як відеокліпи та фільми.

Кожна модальність представляє унікальні виклики та можливості для мовних моделей. Наприклад, зображення вимагають від моделі розуміння візуальних концепцій та зв'язків, тоді як аудіо вимагає від моделі обробки та генерації мовлення та інших звуків.

Мультимодальні Мовні Моделі

Мультимодальні мовні моделі розроблені для роботи з кількома модальностями в межах однієї моделі. Ці моделі зазвичай мають спеціалізовані компоненти або шари, які можуть як розуміти вхідні дані, так і генерувати вихідні дані в різних модальностях. Деякі помітні приклади мультимодальних мовних моделей включають:

- **OpenAI's GPT-4o**: GPT-4o — це велика мовна модель, яка природно розуміє та обробляє мовленнєве аудіо на додачу до тексту. Ця можливість дозволяє GPT-4o виконувати такі завдання, як транскрибування усного мовлення, генерація тексту з аудіовходів та надання відповідей на основі усних запитів.
- **OpenAI's GPT-4 з візуальним входом**: GPT-4 — це велика мовна модель, яка може обробляти як текст, так і зображення. Коли їй надається зображення як вхідні дані, GPT-4 може аналізувати вміст зображення та генерувати текст, який описує або відповідає на візуальну інформацію.

- **Google's Gemini**: Gemini — це мультимодальна модель, яка може працювати з текстом, зображеннями та відео. Вона використовує уніфіковану архітектуру, яка дозволяє крос-модальне розуміння та генерацію, уможливлюючи такі завдання, як генерація підписів до зображень, узагальнення відео та відповіді на запитання на основі зображень.

- **DALL-E та Stable Diffusion**: Хоча вони не є мовними моделями в традиційному розумінні, ці моделі демонструють потужність мультимодального ШІ, генеруючи зображення з текстових описів. Вони показують потенціал моделей, які можуть здійснювати переклад між різними модальностями.

Переваги та застосування мультимодальних моделей

Мультимодальні мовні моделі пропонують кілька переваг і уможливлюють широкий спектр застосувань, включаючи:

- **Покращене розуміння**: Обробляючи інформацію з кількох модальностей, ці моделі можуть отримати більш всебічне розуміння світу, подібно до того, як люди навчаються через різні сенсорні входи.
- **Кросмодальна генерація**: Мультимодальні моделі можуть генерувати контент в одній модальності на основі вхідних даних з іншої, наприклад, створювати зображення з текстового опису або генерувати відеорезюме з написаної статті.
- **Доступність**: Мультимодальні моделі можуть зробити інформацію більш доступною шляхом перекладу між модальностями, наприклад, генеруючи текстові описи зображень для користувачів з вадами зору або створюючи аудіоверсії письмового контенту.
- **Творчі застосування**: Мультимодальні моделі можна використовувати для творчих завдань, таких як генерація мистецтва, музики або відео на основі

текстових підказок, відкриваючи нові можливості для митців та творців контенту.

Оскільки мультимодальні мовні моделі продовжують розвиватися, вони, ймовірно, відіграватимуть все важливішу роль у розробці застосунків на базі ШІ, які можуть розуміти та генерувати контент у різних модальностях. Це уможливить більш природну та інтуїтивну взаємодію між людьми та системами ШІ, а також відкриє нові можливості для творчого вираження та поширення знань.

Екосистеми провайдерів

Коли йдеться про інтеграцію великих мовних моделей (ВММ) у застосунки, ви маєте зростаючий вибір опцій. Кожен великий провайдер ВММ, такий як OpenAI, Anthropic, Google та Cohere, пропонує власну екосистему моделей, API та інструментів. Вибір правильного провайдера включає розгляд різних факторів, включаючи ціноутворення, продуктивність, фільтрацію контенту, конфіденційність даних та опції налаштування.

OpenAI

OpenAI є одним з найвідоміших провайдерів ВММ, їхня серія GPT (GPT-3, GPT-4) широко використовується в різних застосунках. OpenAI пропонує зручний API, який дозволяє легко інтегрувати їхні моделі в застосунки. Вони надають ряд моделей з різними можливостями та ціновими категоріями, від базової моделі Ada до потужної моделі Davinci.

Екосистема OpenAI також включає інструменти, такі як OpenAI Playground, який дозволяє експериментувати з промптами та точно налаштовувати моделі для конкретних випадків використання. Вони пропонують опції фільтрації контенту, щоб запобігти генерації неприйнятного або шкідливого вмісту.

Для безпосередньої роботи з моделями OpenAI я використовую бібліотеку ruby-openai від Alex Rudall.

Anthropic

Anthropic є ще одним важливим гравцем у сфері ВММ, їхні моделі Claude набувають популярності завдяки високій продуктивності та етичним міркуванням. Anthropic зосереджується на розробці безпечних та відповідальних систем ШІ, з сильним акцентом на фільтрації контенту та уникненні шкідливих результатів.

Екосистема Anthropic включає API Claude, який дозволяє інтегрувати модель у їхні застосунки, а також інструменти для інженерії промптів та тонкого налаштування. Вони також пропонують модель Claude Instant, яка включає можливості веб-пошуку для отримання більш актуальних та фактичних відповідей.

Для безпосередньої роботи з моделями Anthropic я використовую бібліотеку anthrophic від Alex Rudall.

Google

Google розробив кілька потужних ВММ, включаючи Gemini, BERT, T5 та PaLM. Ці моделі відомі своєю високою продуктивністю в широкому спектрі завдань з обробки природної мови. Екосистема Google включає бібліотеки TensorFlow та Keras, які надають інструменти та фреймворки для побудови та навчання моделей машинного навчання.

Google також пропонує Cloud AI Platform, яка дозволяє легко розгортати та масштабувати їхні моделі в хмарі. Вони надають ряд попередньо навчених моделей та API для завдань, таких як аналіз тональності, розпізнавання сутностей та переклад.

Meta

Meta, раніше відома як Facebook, активно інвестує в розробку великих мовних моделей, що підкреслюється випуском таких моделей, як LLaMA та OPT. Ці моделі виділяються своєю високою продуктивністю в різноманітних мовних завданнях і здебільшого доступні через канали з відкритим кодом, підтримуючи прихильність Meta до досліджень та співпраці з спільнотою.

Екосистема Meta в основному побудована навколо PyTorch, бібліотеки машинного навчання з відкритим кодом, яка цінується за свої динамічні обчислювальні можливості та гнучкість, що сприяє інноваційним дослідженням та розробці ШІ.

Окрім своїх технічних пропозицій, Meta приділяє значну увагу етичному розвитку ШІ. Вони впроваджують надійну фільтрацію контенту та зосереджуються на зменшенні упереджень, що відповідає їхнім ширшим цілям безпеки та відповідальності у застосуванні ШІ.

Cohere

Cohere є новішим гравцем у сфері ВММ, що зосереджується на тому, щоб зробити ВММ доступнішими та простішими у використанні порівняно з конкурентами. Їхня екосистема включає Cohere API, який надає доступ до ряду попередньо навчених моделей для таких завдань, як генерація тексту, класифікація та узагальнення.

Cohere також пропонує інструменти для інженерії промптів, точного налаштування та фільтрації контенту. Вони приділяють особливу увагу конфіденційності та безпеці даних, пропонуючи такі функції, як шифроване зберігання даних та контроль доступу.

Ollama

Ollama - це платформа для самостійного хостингу, яка дозволяє користувачам керувати та розгортати різні великі мовні моделі (ВММ) локально на своїх машинах, надаючи їм повний контроль над моделями ШІ без залежності від зовнішніх хмарних сервісів. Така конфігурація ідеально підходить для тих, хто надає пріоритет конфіденційності даних і бажає керувати своїми операціями зі ШІ власноруч.

Платформа підтримує ряд моделей, включаючи версії Llama, Phi, Gemma та Mistral, які відрізняються за розміром та обчислювальними вимогами. Ollama спрощує завантаження та запуск цих моделей безпосередньо з командного рядка за допомогою простих команд на кшталт ollama run <model_name>, і вона розроблена для роботи на різних операційних системах, включаючи macOS, Linux та Windows.

Для розробників, які прагнуть інтегрувати моделі з відкритим кодом у свої додатки без використання віддаленого API, Ollama пропонує CLI для керування життєвим циклом моделей, подібно до інструментів керування контейнерами. Він також підтримує користувацькі конфігурації та промпти, що дозволяє досягти високого ступеня налаштування для адаптації моделей під конкретні потреби чи випадки використання.

Ollama особливо підходить для технічно обізнаних користувачів та розробників завдяки своєму інтерфейсу командного рядка та гнучкості в керуванні та розгортанні моделей ШІ. Це робить її потужним інструментом для бізнесу та окремих осіб, які потребують надійних можливостей ШІ без компромісів щодо безпеки та контролю.

Мультимодельні платформи

Крім того, існують провайдери, які розміщують широкий спектр моделей з відкритим кодом, такі як Together.ai та Groq.. Ці платформи пропонують

гнучкість та можливість налаштування, дозволяючи запускати і, в деяких випадках, навіть точно налаштовувати моделі з відкритим кодом відповідно до ваших конкретних потреб. Наприклад, Together.ai надає доступ до ряду ВММ з відкритим кодом, що дозволяє користувачам експериментувати з різними моделями та конфігураціями. Groq зосереджується на наданні надвисокопродуктивного завершення, яке на момент написання цієї книги здається майже магічним

Вибір постачальника ВММ

При виборі постачальника ВММ слід враховувати такі фактори:

- **Ціноутворення**: Різні постачальники пропонують різні моделі ціноутворення, від оплати за використання до планів на основі підписки. Важливо враховувати очікуване використання та бюджет при виборі постачальника.
- **Продуктивність**: Продуктивність ВММ може суттєво відрізнятися між постачальниками, тому важливо провести тестування моделей на конкретних випадках використання перед прийняттям рішення.
- **Фільтрація контенту**: Залежно від застосування, фільтрація контенту може бути критично важливим фактором. Деякі постачальники пропонують більш надійні опції фільтрації контенту, ніж інші.
- **Конфіденційність даних**: Якщо додаток обробляє конфіденційні дані користувачів, важливо вибрати постачальника з надійними практиками конфіденційності та безпеки даних.
- **Налаштування**: Деякі постачальники пропонують більшу гнучкість щодо точного налаштування та адаптації моделей для конкретних випадків використання.

Зрештою, вибір постачальника ВММ залежить від конкретних вимог та обмежень застосування. Ретельно оцінюючи варіанти та враховуючи такі фактори, як

ціноутворення, продуктивність та конфіденційність даних, ви можете вибрати постачальника, який найкраще відповідає вашим потребам.

Також варто зазначити, що ландшафт ВММ постійно розвивається, регулярно з'являються нові постачальники та моделі. Варто слідкувати за останніми розробками та бути відкритим до вивчення нових можливостей, коли вони стають доступними.

OpenRouter

Протягом цієї книги я буду покладатися виключно на OpenRouter як на мій вибраний API-провайдер. Причина проста: це універсальний магазин для всіх найпопулярніших комерційних моделей та моделей з відкритим кодом. Якщо ви прагнете попрактикуватися з програмуванням ШІ, одним з найкращих місць для початку є моя власна OpenRouter Ruby Library.

Міркування про продуктивність

При інтеграції мовних моделей у додатки, продуктивність є критично важливим фактором. Продуктивність мовної моделі можна виміряти з точки зору її *затримки* (час, необхідний для генерації відповіді) та *пропускної здатності* (кількість запитів, які вона може обробити за одиницю часу).

Час до першого токену (ЧДПТ) є ще одним важливим показником продуктивності, особливо актуальним для чатботів та додатків, що потребують інтерактивних відповідей у реальному часі. ЧДПТ вимірює затримку від моменту отримання запиту користувача до моменту генерації першого слова (або токену) відповіді. Цей показник є вирішальним для підтримки безперебійного та захоплюючого користувацького досвіду, оскільки затримки у відповідях можуть призвести до розчарування користувачів та втрати їхньої зацікавленості.

Ці показники продуктивності можуть мати значний вплив на користувацький досвід та масштабованість додатку.

Декілька ƒакторів можуть впливати на продуктивність мовної моделі, включаючи:

Кількість параметрів: Більші моделі з більшою кількістю параметрів зазвичай потребують більше обчислювальних ресурсів і можуть мати вищу затримку та нижчу пропускну здатність порівняно з меншими моделями.

Апаратне забезпечення: Продуктивність мовної моделі може значно відрізнятися залежно від апаратного забезпечення, на якому вона працює. Хмарні провайдери пропонують екземпляри GPU та TPU, оптимізовані для машинного навчання, які можуть значно прискорити виведення моделі.

Однією з чудових особливостей OpenRouter є те, що для багатьох запропонованих моделей ви отримуєте вибір хмарних провайдерів з різними профілями продуктивності та вартості.

Квантизація: Методи квантизації можна використовувати для зменшення обсягу пам'яті та обчислювальних вимог моделі шляхом представлення ваг та активацій типами даних нижчої точності. Це може покращити продуктивність без значного погіршення якості. Як розробник додатків, ви, ймовірно, не будете займатися навчанням власних моделей на різних рівнях квантизації, але добре хоча б бути знайомим з термінологією.

Пакетна обробка: Обробка кількох запитів одночасно в пакетах може покращити пропускну здатність за рахунок амортизації накладних витрат на завантаження моделі та передачу даних.

Кешування: Кешування результатів часто використовуваних промптів або вхідних послідовностей може зменшити кількість запитів на виведення та покращити загальну продуктивність.

При виборі мовної моделі для виробничого додатку важливо провести тестування її продуктивності на репрезентативних робочих навантаженнях та конфігураціях обладнання. Це може допомогти виявити потенційні вузькі місця та забезпечити відповідність моделі необхідним показникам продуктивності.

Також варто враховувати компроміси між продуктивністю моделі та іншими факторами, такими як вартість, гнучкість та простота інтеграції. Наприклад, використання меншої, дешевшої моделі з нижчою затримкою може бути кращим для додатків, що потребують відповідей у реальному часі, тоді як більша, потужніша модель може краще підходити для пакетної обробки або складних завдань міркування.

Експерименти з різними моделями ВММ

Вибір ВММ рідко є остаточним рішенням. Оскільки нові та вдосконалені моделі випускаються регулярно, добре будувати додатки модульним способом, який дозволяє з часом замінювати різні мовні моделі. Промпти та набори даних часто можна повторно використовувати для різних моделей з мінімальними змінами. Це дозволяє використовувати найновіші досягнення в моделюванні мови без необхідності повністю переробляти додатки.

Можливість легко перемикатися між широким спектром моделей - це ще одна причина, чому я люблю OpenRouter.

При переході на нову мовну модель важливо ретельно протестувати та перевірити її продуктивність та якість виводу, щоб переконатися, що вона відповідає вимогам додатку. Це може включати перенавчання або точне налаштування моделі на специфічних даних домену, а також оновлення будь-яких залежних компонентів, що використовують виводи моделі.

Проектуючи додатки з урахуванням продуктивності та модульності, ви можете створювати масштабовані, ефективні та перспективні системи, які можуть адаптуватися до ландшафту технологій моделювання мови, що швидко розвивається.

Складені системи ШІ

Перед завершенням нашого вступу варто згадати, що до 2023 року та вибуху інтересу до генеративного ШІ, спричиненого ChatGPT, традиційні підходи до ШІ зазвичай покладалися на інтеграцію окремих, закритих моделей. На противагу цьому, *Складені системи ШІ* використовують складні конвеєри взаємопов'язаних компонентів, що працюють разом для досягнення інтелектуальної поведінки.

У своїй основі складені системи ШІ складаються з множини модулів, кожен з яких призначений для виконання конкретних завдань або функцій. Ці модулі можуть включати генератори, пошуковики, ранжувальники, класифікатори та різні інші спеціалізовані компоненти. Розбиваючи загальну систему на менші, сфокусовані блоки, розробники можуть створювати більш гнучкі, масштабовані та підтримувані архітектури ШІ.

Однією з ключових переваг складених систем ШІ є їхня здатність поєднувати сильні сторони різних технік та моделей ШІ. Наприклад, система може використовувати велику мовну модель (ВММ) для розуміння та генерації природної мови, одночасно застосовуючи окрему модель для пошуку інформації чи прийняття рішень на основі правил. Такий модульний підхід дозволяє вибирати найкращі інструменти та методи для кожного конкретного завдання, замість того, щоб покладатися на універсальне рішення.

Проте, створення складених систем ШІ також створює унікальні виклики. Зокрема, забезпечення загальної узгодженості та послідовності поведінки системи вимагає надійних механізмів тестування, моніторингу та управління.

Поява потужних ВММ, таких як GPT-4, дозволяє нам експериментувати зі складеними системами ШІ легше, ніж будь-коли раніше, оскільки ці передові моделі здатні виконувати множинні ролі в складеній системі, такі як класифікація, ранжування та генерація, на додаток до їхніх можливостей розуміння природної мови. Ця універсальність дозволяє розробникам швидко створювати прототипи та ітерувати архітектури складених ШІ, відкриваючи нові можливості для розробки інтелектуальних застосунків.

Патерни розгортання для складених систем ШІ

Складені системи ШІ можуть бути розгорнуті за допомогою різних патернів, кожен з яких розроблений для вирішення конкретних вимог та випадків використання. Розглянемо чотири поширені патерни розгортання: Запитання та Відповіді, Багатоагентні/Агентні Розв'язувачі Задач, Розмовний ШІ та Копілоти.

Запитання та Відповіді

Системи запитань та відповідей (Q&A) зосереджені на наданні пошуку інформації, покращеного можливостями розуміння моделей ШІ, щоб функціонувати як щось більше, ніж просто пошукова система. Поєднуючи потужні мовні моделі із зовнішніми джерелами знань за допомогою генерації з підкріпленням вибіркою (RAG), системи запитань та відповідей уникають галюцинацій та надають точні й контекстуально релевантні відповіді на запити користувачів.

Ключові компоненти системи Q&A на основі ВММ включають:

- **Розуміння та переформулювання запиту**: Аналіз запитів користувачів та їх переформулювання для кращої відповідності базовим джерелам знань.

- **Пошук знань**: Отримання релевантної інформації зі структурованих або неструктурованих джерел даних на основі переформульованого запиту.
- **Генерація відповіді**: Генерація зв'язних та інформативних відповідей шляхом інтеграції отриманих знань із генеративними можливостями мовної моделі.

Підсистеми RAG особливо важливі в областях Q&A, де надання точної та актуальної інформації є критичним, таких як підтримка клієнтів, управління знаннями або освітні застосунки

Багатоагентні/Агентні Розв'язувачі Задач

Багатоагентні, також відомі як *Агентні*, системи складаються з множини автономних агентів, що працюють разом для вирішення складних задач. Кожен агент має конкретну роль, набір навичок та доступ до відповідних інструментів чи джерел інформації. Співпрацюючи та обмінюючись інформацією, ці агенти можуть вирішувати завдання, які було б складно або неможливо виконати одному агенту.

Ключові принципи багатоагентних розв'язувачів задач включають:

- **Спеціалізація**: Кожен агент зосереджується на конкретному аспекті проблеми, використовуючи свої унікальні можливості та знання.
- **Співпраця**: Агенти спілкуються та координують свої дії для досягнення спільної мети, часто через передачу повідомлень або спільну пам'ять.
- **Адаптивність**: Система може адаптуватися до змінних умов або вимог шляхом коригування ролей та поведінки окремих агентів.

Багатоагентні системи добре підходять для застосунків, що вимагають розподіленого вирішення проблем, таких як оптимізація ланцюгів постачання, керування трафіком або планування реагування на надзвичайні ситуації

Розмовний ШІ

Системи розмовного ШІ забезпечують взаємодію природною мовою між користувачами та інтелектуальними агентами. Ці системи поєднують можливості розуміння природної мови, керування діалогом та генерації мови для забезпечення захоплюючого та персоналізованого розмовного досвіду.

Основні компоненти системи розмовного ШІ включають:

- **Розпізнавання намірів**: Визначення наміру користувача на основі його введення, наприклад, запитання, запиту або вираження почуттів.
- **Витяг сутностей**: Вилучення релевантних сутностей або параметрів із введення користувача, таких як дати, місця розташування або назви продуктів.
- **Керування діалогом**: Підтримка стану розмови, визначення відповідної реакції на основі наміру користувача та контексту, та обробка багатокрокових взаємодій.
- **Генерація відповіді**: Генерація людиноподібних відповідей за допомогою мовних моделей, шаблонів або методів на основі пошуку.

Системи розмовного ШІ зазвичай використовуються в чат-ботах для обслуговування клієнтів, віртуальних помічниках та голосових інтерфейсах. Як згадувалося раніше, більшість підходів, патернів та прикладів коду в цій книзі безпосередньо взяті з моєї роботи над великою системою розмовного ШІ під назвою Olympia

Копілоти

Копілоти — це помічники на базі ШІ, які працюють пліч-о-пліч з користувачами для підвищення їхньої продуктивності та покращення процесу прийняття рішень. Ці системи використовують комбінацію обробки природної мови,

машинного навчання та галузевих знань для надання інтелектуальних рекомендацій, автоматизації завдань та контекстуальної підтримки.

Ключові особливості Копілотів включають:

- **Персоналізація**: Адаптація до індивідуальних уподобань користувача, робочих процесів та стилів спілкування.
- **Проактивна допомога**: Передбачення потреб користувача та надання відповідних пропозицій чи дій без явних запитів.
- **Постійне навчання**: Покращення продуктивності з часом через навчання на основі відгуків користувачів, взаємодій та даних.

Копілоти все частіше використовуються в різних галузях, таких як розробка програмного забезпечення (наприклад, автодоповнення коду та виявлення помилок), творче письмо (наприклад, пропозиції щодо контенту та редагування), та аналіз даних (наприклад, інсайти та рекомендації щодо візуалізації)

Ці шаблони розгортання демонструють універсальність та потенціал складених систем ШІ. Розуміючи характеристики та випадки використання кожного шаблону, ви можете приймати обґрунтовані рішення при проектуванні та впровадженні інтелектуальних застосунків. Хоча ця книга не присвячена конкретно реалізації складених систем ШІ, багато, якщо не всі, з тих самих підходів та шаблонів застосовуються для інтеграції дискретних компонентів ШІ в межах традиційної розробки застосунків.

Ролі в складених системах ШІ

Складені системи ШІ побудовані на основі взаємопов'язаних модулів, кожен з яких призначений для виконання конкретної ролі. Ці модулі працюють разом для створення інтелектуальної поведінки та вирішення складних проблем. Корисно бути знайомим з цими ролями, коли ви думаєте про те, де можна впровадити чи замінити частини вашого застосунку дискретними компонентами ШІ.

Генератор

Генератори відповідають за створення нових даних або контенту на основі вивчених шаблонів чи вхідних запитів. У світі ШІ існує багато різних видів генераторів, але в контексті мовних моделей, які представлені в цій книзі, генератори можуть створювати текст, подібний до людського, завершувати часткові речення або генерувати відповіді на запити користувачів. Вони відіграють важливу роль у таких завданнях, як створення контенту, генерація діалогів та розширення даних.

Пошуковик

Пошуковики використовуються для пошуку та вилучення релевантної інформації з великих наборів даних або баз знань. Вони застосовують такі методи, як семантичний пошук, співставлення ключових слів або векторну подібність для знаходження найбільш відповідних точок даних на основі заданого запиту чи контексту. Пошуковики є важливими для завдань, які вимагають швидкого доступу до конкретної інформації, таких як відповіді на запитання, перевірка фактів або рекомендації контенту.

Ранжувальник

Ранжувальники відповідають за впорядкування або визначення пріоритетності набору елементів на основі певних критеріїв чи показників релевантності. Вони призначають ваги або бали кожному елементу, а потім сортують їх відповідно. Ранжувальники зазвичай використовуються в пошукових системах, системах рекомендацій або будь-яких застосунках, де важливо представити користувачам найбільш релевантні результати.

Класифікатор

Класифікатори використовуються для категоризації або маркування точок даних на основі попередньо визначених класів або категорій. Вони навчаються на розмічених тренувальних даних, а потім передбачають клас нових, невідомих прикладів. Класифікатори є фундаментальними для таких завдань, як аналіз настроїв, виявлення спаму або розпізнавання зображень, де метою є призначення конкретної категорії кожному вхідному елементу.

Інструменти та Агенти

На додаток до цих основних ролей, складені системи ШІ часто включають інструменти та агентів для розширення їхньої функціональності та адаптивності:

- **Інструменти**: Інструменти — це дискретні програмні компоненти або API, які виконують конкретні дії чи обчислення. Вони можуть викликатися іншими модулями, такими як генератори або пошуковики, для виконання підзадач або збору додаткової інформації. Прикладами інструментів є пошукові системи, калькулятори або бібліотеки візуалізації даних.
- **Агенти**: Агенти — це автономні сутності, які можуть сприймати своє середовище, приймати рішення та виконувати дії для досягнення конкретних цілей. Вони часто покладаються на комбінацію різних методів ШІ, таких як планування, міркування та навчання, щоб ефективно працювати в динамічних або невизначених умовах. Агенти можуть використовуватися для моделювання складної поведінки або для координації дій кількох модулів у складеній системі ШІ.

У чистій складеній системі ШІ взаємодія між цими компонентами оркеструється через чітко визначені інтерфейси та протоколи комунікації. Дані протікають між модулями, де вихід одного компонента служить входом для іншого. Така модульна архітектура забезпечує гнучкість, масштабованість та легкість

обслуговування, оскільки окремі компоненти можна оновлювати, замінювати або розширювати, не впливаючи на всю систему.

Використовуючи потужність цих компонентів та їхню взаємодію, складені системи ШІ можуть вирішувати складні реальні проблеми, які вимагають комбінації різних можливостей ШІ. Досліджуючи підходи та шаблони для інтеграції ШІ в розробку застосунків, пам'ятайте, що ті самі принципи та методи, що використовуються в складених системах ШІ, можна застосувати для створення інтелектуальних, адаптивних та орієнтованих на користувача застосунків.

У наступних розділах Частини 1 ми глибше зануримося в фундаментальні підходи та методи інтеграції компонентів ШІ у ваш процес розробки застосунків. Від інженерії промптів та генерації з підсиленим пошуком до самовідновлюваних даних та інтелектуальної оркестрації робочих процесів, ми охопимо широкий спектр шаблонів та найкращих практик, щоб допомогти вам створювати передові застосунки на базі ШІ.

Частина 1: Фундаментальні підходи та методики

Ця частина книги представляє різні способи інтеграції ШІ у ваші застосунки. Розділи охоплюють низку пов'язаних підходів та методик, починаючи від більш високорівневих концепцій, таких як Звуження шляху та Генерація з розширеним пошуком, і закінчуючи ідеями щодо програмування власного рівня абстракції поверх API завершення чату ВММ.

Мета цієї частини книги — допомогти вам зрозуміти види поведінки, які ви можете реалізувати за допомогою ШІ, перш ніж заглиблюватися у конкретні шаблони реалізації, які є основною темою Частини 2.

Підходи в Частині 1 базуються на ідеях, які я використовував у своєму коді, класичних патернах архітектури корпоративних застосунків та інтеграції, а також метафорах, які я використовував, пояснюючи можливості ШІ іншим людям, включаючи нетехнічних бізнес-стейкхолдерів.

Звузити шлях

“Звузити шлях” означає зосередження ШІ на конкретному завданні. Я використовую це як мантру щоразу, коли починаю дратуватися через те, що ШІ поводиться “по-дурному” або неочікувано. Ця мантра нагадує мені, що невдача, ймовірно, моя провина, і що мені, мабуть, варто ще більше звузити шлях.

Потреба у звуженні шляху виникає через величезний обсяг знань, що містяться у великих мовних моделях, особливо у моделях світового класу, таких як від OpenAI та Anthropic, які мають буквально трильйони параметрів.

Доступ до такого широкого спектру знань, безсумнівно, є потужним і породжує емерджентну поведінку, таку як теорія свідомості та здатність міркувати подібно до людини. Проте, цей приголомшливий обсяг інформації також створює проблеми, коли йдеться про генерування точних і правильних відповідей на конкретні промпти, особливо якщо ці промпти мають демонструвати детерміновану поведінку, яку можна інтегрувати зі "звичайною" розробкою програмного забезпечення та алгоритмами.

Низка факторів призводить до цих проблем.

Інформаційне перевантаження: Великі мовні моделі навчаються на масивних обсягах даних, що охоплюють різні домени, джерела та часові періоди. Ці обширні знання дозволяють їм займатися різноманітними темами та генерувати відповіді на основі широкого розуміння світу. Однак, коли модель стикається з конкретним промптом, вона може мати труднощі з фільтрацією нерелевантної, суперечливої або застарілої/неактуальної інформації, що призводить до відповідей, яким бракує фокусу чи точності. Залежно від того, що ви намагаєтесь зробити, сам обсяг *суперечливої* інформації, доступної моделі, може легко перевищити її здатність надати відповідь або поведінку, яку ви шукаєте.

Контекстна неоднозначність: Враховуючи величезний *прихований простір* знань, великі мовні моделі можуть зіткнутися з неоднозначністю при спробі зрозуміти *контекст* вашого промпту. Без належного звуження чи керування модель може генерувати відповіді, які лише дотично пов'язані, але не відповідають безпосередньо вашим намірам. Такий тип невдачі призводить до відповідей, які не відповідають темі, непослідовні або не відповідають вашим заявленим потребам. У цьому випадку звуження шляху означає *усунення неоднозначності* контексту, забезпечуючи, щоб наданий вами контекст змушував модель зосереджуватися лише на найбільш релевантній інформації в її базових знаннях.

Примітка: Коли ви тільки починаєте займатися “інженерією промптів”, ви, швидше за все, проситимете модель робити щось без належного пояснення бажаного результату; потрібна практика, щоб не бути неоднозначним!

Часові невідповідності: Оскільки мовні моделі навчаються на даних, створених у різні часові періоди, вони можуть містити знання, які застаріли, були замінені або більше не є точними. Наприклад, інформація про поточні події, наукові відкриття чи технологічні досягнення могла еволюціонувати з моменту збору тренувальних даних моделі. Без звуження шляху для надання пріоритету більш новим та надійним джерелам, модель може генерувати відповіді на основі застарілої або неправильної інформації, що призводить до неточностей та невідповідностей у її виводах.

Особливості конкретних доменів: Різні домени та галузі мають свою специфічну термінологію, конвенції та бази знань. Подумайте про практично будь-який TLA (Трилітерний Акронім) і ви зрозумієте, що більшість із них мають більше одного значення. Наприклад, MSK може означати Amazon’s Managed Streaming for Apache Kafka, Memorial Sloan Kettering Cancer Center, або людську опорно-рухову систему (MusculoSKeletal).

Коли промпт вимагає експертизи в конкретній області, загальних знань великої мовної моделі може бути недостатньо для надання точних і нюансованих відповідей. Звуження шляху шляхом фокусування на доменно-специфічній інформації, або через інженерію промптів, або через генерацію з розширеним пошуком, дозволяє моделі генерувати відповіді, які краще відповідають вимогам та очікуванням вашого конкретного домену.

Прихований простір: Незбагненно величезний

Коли я згадую “прихований простір” мовної моделі, я маю на увазі величезний багатовимірний ландшафт знань та інформації, який модель засвоїла під час процесу навчання. Це як прихований світ всередині нейронних мереж моделі, де зберігаються всі патерни, асоціації та представлення мови.

Уявіть, що ви досліджуєте величезну, недосліджену територію, заповнену незліченними взаємопов’язаними вузлами. Кожен вузол представляє частину інформації, концепцію або зв’язок, який вивчила модель. Коли ви переміщаєтеся цим простором, ви побачите, що деякі вузли розташовані ближче один до одного, що вказує на сильний зв’язок або подібність, тоді як інші знаходяться далі, що свідчить про слабший або віддаленіший зв’язок.

Складність прихованого простору полягає в тому, що він неймовірно складний і багатовимірний. Уявіть собі щось настільки ж величезне, як наш фізичний всесвіт, з його скупченнями галактик та величезними, неосяжними відстанями порожнього простору між ними.

Оскільки він містить тисячі вимірів, прихований простір неможливо безпосередньо спостерігати чи інтерпретувати людям. Це абстрактне представлення, яке модель використовує внутрішньо для обробки та генерації мови. Коли ви надаєте моделі вхідний запит, вона по суті відображає цей запит на конкретне місце в прихованому просторі. Потім модель використовує навколишню інформацію та зв’язки в цьому просторі для генерації відповіді.

Річ у тім, що модель засвоїла величезний обсяг інформації зі своїх навчальних даних, і не вся вона є релевантною чи точною для конкретного завдання. Ось чому звуження шляху стає таким важливим. Надаючи чіткі інструкції, приклади та контекст у ваших запитах, ви по суті спрямовуєте модель зосередитися на конкретних областях у прихованому просторі, які найбільш релевантні для бажаного результату.

Інший спосіб уявити це - як використання прожектора в повністю темному музеї. Якщо ви коли-небудь відвідували Лувр або Метрополітен-музей, то це саме той масштаб, про який я говорю. Прихований простір - це музей, наповнений незліченними об’єктами та деталями. Ваш запит - це прожектор, що освітлює конкретні області та привертає увагу моделі до найважливішої інформації. Без такого керування модель може безцільно блукати прихованим простором, збираючи неважливу або суперечливу інформацію по дорозі.

Працюючи з мовними моделями та створюючи свої запити, пам’ятайте про концепцію прихованого простору. Ваша мета - ефективно орієнтуватися в цьому величезному ландшафті знань, спрямовуючи модель до найбільш релевантної та точної інформації для вашого завдання. Звужуючи шлях та надаючи чіткі вказівки, ви можете розкрити повний потенціал прихованого простору моделі та генерувати якісні, послідовні відповіді.

Хоча попередні описи мовних моделей та прихованого простору, яким вони керуються, можуть здаватися дещо магічними чи абстрактними, важливо розуміти, що запити - це не заклинання чи магічні ƒормули. Принцип роботи мовних моделей базується на засадах лінійної алгебри та теорії ймовірностей.

У своїй основі мовні моделі - це імовірнісні моделі тексту, подібно до того, як крива нормального розподілу є статистичною моделлю даних. Вони навчаються через процес, який називається авторегресійним моделюванням, де модель вчиться передбачати ймовірність наступного слова в послідовності на основі слів, що йдуть перед ним. Під час навчання модель починає з випадкових ваг і поступово коригує їх, щоб призначати вищі ймовірності тексту, який нагадує реальні зразки, на яких вона навчалася.

Однак, думати про мовні моделі як про прості статистичні моделі, такі як лінійна регресія, не дає найкращої інтуїції для розуміння їхньої поведінки. Більш влучна аналогія - розглядати їх як імовірнісні програми, які є моделями, що дозволяють маніпулювати випадковими змінними і можуть представляти

складні статистичні взаємозв'язки.

Імовірнісні програми можна представити за допомогою графічних моделей, які надають візуальний спосіб розуміння залежностей та взаємозв'язків між змінними в моделі. Ця перспектива може запропонувати цінні уявлення про роботу складних моделей генерації тексту, таких як GPT-4 та Claude.

У статті "Language Model Cascades" авторства Дохана та ін., автори заглиблюються в деталі того, як імовірнісні програми можна застосовувати до мовних моделей. Вони показують, як цю структуру можна використовувати для розуміння поведінки цих моделей та спрямування розробки більш ефективних стратегій формування запитів.

Один ключовий висновок з цієї імовірнісної перспективи полягає в тому, що мовна модель по суті створює портал до альтернативного всесвіту, де існують бажані документи. Модель призначає ваги всім можливим документам на основі їхньої ймовірності, ефективно звужуючи простір можливостей, щоб зосередитися на найбільш релевантних.

Це повертає нас до центральної теми "звуження шляху". Основна мета формування запитів полягає в тому, щоб обумовити імовірнісну модель таким чином, щоб сфокусувати масу її передбачень, зосередившись на конкретній інформації чи поведінці, яку ми хочемо отримати. Надаючи ретельно складені запити, ми можемо спрямовувати модель ефективніше орієнтуватися в прихованому просторі та генерувати більш релевантні та послідовні результати.

Однак важливо пам'ятати, що мовна модель в кінцевому підсумку обмежена інформацією, на якій вона навчалася. Хоча вона може генерувати текст, подібний до існуючих документів, або комбінувати ідеї по-новому, вона не може створювати повністю нову інформацію з нічого. Наприклад, ми не можемо очікувати, що модель надасть ліки від раку, якщо такі ліки ще не були винайдені та задокументовані в її навчальних даних.

Натомість сила моделі полягає в її здатності знаходити та синтезувати

інформацію, подібну до тієї, що міститься в наших промптах. Розуміючи імовірнісну природу цих моделей і те, як промпти можна використовувати для обумовлення їхніх результатів, ми можемо ефективніше використовувати їхні можливості для генерування цінних висновків та контенту.

Розгляньмо наведені нижче промпти. У першому випадку саме слово "Mercury" може стосуватися планети, хімічного елемента або римського бога, але найімовірніше йдеться про планету. Справді, GPT-4 надає розлогу відповідь, яка починається зі слів *Меркурій — це найменша та найближча до Сонця планета Сонячної системи...*. Другий промпт конкретно стосується хімічного елемента. Третій стосується постаті з римської міфології, відомої своєю швидкістю та роллю божественного посланця.

```
# Prompt 1
Tell me about: Mercury

# Prompt 2
Tell me about: Mercury element

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

Додавши лише кілька додаткових слів, ми повністю змінили реакцію ШІ. Як ви дізнаєтеся далі в книзі, такі вишукані прийоми інженерії промптів як промптування з n-прикладами, структурований ввід/вивід та Ланцюжок Міркувань - це просто розумні способи обумовлення виводу моделі.

Отже, зрештою, мистецтво інженерії промптів полягає в розумінні того, як орієнтуватися у величезному імовірнісному ландшафті знань мовної моделі, щоб звузити шлях до конкретної інформації чи поведінки, яку ми шукаємо.

Для читачів із ґрунтовним розумінням вищої математики, базування вашого розуміння цих моделей на принципах теорії ймовірностей та лінійної алгебри може безумовно допомогти! Для решти з вас, хто хоче розробити ефективні

стратегії для отримання бажаних результатів, давайте триматися більш інтуїтивних підходів.

Як "Звужується" Шлях

Щоб подолати ці виклики надмірних знань, ми використовуємо методи, які допомагають спрямовувати процес генерації мовної моделі та фокусувати її увагу на найбільш релевантній та точній інформації.

Ось найважливіші методи у рекомендованому порядку, тобто спочатку варто спробувати Інженерію Промптів, потім RAG, і нарешті, якщо це необхідно, точне налаштування.

Інженерія Промптів Найфундаментальніший підхід полягає у створенні промптів, які включають конкретні інструкції, обмеження чи приклади для спрямування генерації відповідей моделлю. Цей розділ охоплює основи Інженерії Промптів у наступній секції, а багато конкретних патернів інженерії промптів ми розглянемо у Частині 2 книги. Ці патерни включають Дистиляцію Промптів, техніку, яка зосереджується на вдосконаленні та оптимізації промптів для вилучення того, що ШІ вважає найбільш релевантною та лаконічною інформацією.

Доповнення Контексту. Динамічне отримання релевантної інформації із зовнішніх баз знань або документів для надання моделі сфокусованого контексту в момент подачі промпту. Популярні техніки доповнення контексту включають Генерацію з Пошуковим Доповненням (RAG). Так звані "онлайн-моделі", як-от ті, що надаються Perplexity, здатні доповнювати свій контекст результатами пошуку в інтернеті в реальному часі.

Незважаючи на свою потужність, LLM не навчені на ваших унікальних наборах даних, які можуть бути приватними або специфічними для проблеми, яку ви намагаєтеся вирішити. Техніки Доповнення Контексту дозволяють надати LLM доступ до даних за API, в SQL базах даних, або захованих у PDF-файлах та презентаціях.

Точне налаштування або Адаптація до домену Навчання моделі на специфічних для домену наборах даних для спеціалізації її знань та можливостей генерації для конкретного завдання чи галузі.

Зниження Температури

Температура - це *гіперпараметр*, який використовується в трансформер-базованих мовних моделях для контролю випадковості та креативності згенерованого тексту. Це значення між 0 та 1, де нижчі значення роблять вивід більш сфокусованим та детермінованим, тоді як вищі значення роблять його більш різноманітним та непередбачуваним.

Коли температура встановлена на 1, мовна модель генерує текст на основі повного розподілу ймовірностей наступного токена, дозволяючи більш креативні та різноманітні відповіді. Однак це також може призвести до того, що модель генеруватиме текст, який є менш релевантним або зв'язним.

З іншого боку, коли температура встановлена на 0, мовна модель завжди вибирає токен з найвищою ймовірністю, ефективно "звужуючи свій шлях". Майже всі мої ШІ-компоненти використовують температуру, встановлену на 0 або близько до неї, оскільки це призводить до більш сфокусованих та передбачуваних відповідей. Це особливо корисно, коли ви хочете, щоб модель *слідувала інструкціям*, звертала увагу на надані їй функції або просто потребуєте більш точних та релевантних відповідей, ніж ті, які ви отримуєте.

Наприклад, якщо ви створюєте чатбота, який має надавати фактичну інформацію, ви можете захотіти встановити нижче значення температури, щоб забезпечити

більш точні та релевантні відповіді. І навпаки, якщо ви створюєте помічника з креативного письма, ви можете захотіти встановити вищу температуру, щоб заохотити більш різноманітні та творчі результати.

Гіперпараметри: Регулятори та Налаштування Виведення

Працюючи з мовними моделями, ви часто зустрічатимете термін "гіперпараметри". В контексті виведення (тобто, коли ви використовуєте модель для генерації відповідей), гіперпараметри подібні до регуляторів та налаштувань, які ви можете змінювати для контролю поведінки та виводу моделі.

Уявіть це як налаштування параметрів складного механізму. Так само, як ви можете повернути ручку для регулювання температури або перемкнути перемикач для зміни режиму роботи, гіперпараметри дозволяють вам тонко налаштовувати спосіб, яким мовна модель обробляє та генерує текст.

Ось поширені гіперпараметри, з якими ви зіткнетеся під час виведення:

- **Температура**: Як щойно згадувалося, цей параметр контролює випадковість і креативність згенерованого тексту. Вища температура призводить до більш різноманітних і непередбачуваних результатів, тоді як нижча температура дає більш сфокусовані та детерміновані відповіді.

- **Top-p (вибірка ядра)**: Цей параметр контролює вибір найменшого набору токенів, сукупна ймовірність яких перевищує певний поріг (p). Це дозволяє отримувати більш різноманітні результати, зберігаючи при цьому зв'язність.

- **Top-k вибірка**: Ця техніка вибирає *k* найбільш імовірних наступних токенів і перерозподіляє ймовірнісну масу між ними. Це може допомогти запобігти генерації моделлю малоймовірних або нерелевантних токенів.

- **Штрафи за частоту** та **присутність**: Ці параметри штрафують модель за надто часте повторення одних і тих самих слів чи фраз (штраф за частоту) або за генерацію слів, яких немає у вхідному запиті (штраф за присутність). Регулюючи ці значення, ви можете спонукати модель створювати більш різноманітні та релевантні результати.

- **Максимальна довжина**: Цей гіперпараметр встановлює верхню межу кількості токенів (слів або підслів), які модель може згенерувати в одній відповіді. Це допомагає контролювати багатослівність і лаконічність згенерованого тексту.

Експериментуючи з різними налаштуваннями гіперпараметрів, ви помітите, що навіть невеликі коригування можуть суттєво вплинути на результат роботи моделі. Це як налаштування рецепту – щіпка солі чи трохи довший час приготування можуть кардинально змінити кінцеву страву.

Ключовим є розуміння того, як кожен гіперпараметр впливає на поведінку моделі, і знаходження правильного балансу для вашого конкретного завдання. Не бійтеся експериментувати з різними налаштуваннями та спостерігати, як вони впливають на згенерований текст. З часом ви розвинете інтуїтивне розуміння того, які гіперпараметри коригувати і як досягти бажаних результатів.

Поєднуючи використання цих параметрів з інженерією промптів, генерацією з підкріпленням через пошук та точним налаштуванням, ви можете ефективно звузити шлях і спрямувати мовну модель на генерацію більш точних, релевантних і цінних відповідей для конкретного випадку використання.

Базові моделі проти моделей, налаштованих на інструкції

Базові моделі – це необроблені, ненавчені версії LLM. Уявіть їх як чисте полотно, ще не підкориговане спеціальним навчанням для розуміння чи виконання інструкцій. Вони побудовані на основі величезного обсягу даних, на яких їх спочатку навчали, і здатні генерувати широкий спектр результатів. Однак без додаткових шарів точного налаштування на основі інструкцій їхні відповіді можуть бути непередбачуваними і вимагають більш нюансованих, ретельно складених промптів для спрямування їх до бажаного результату. Робота з базовими моделями подібна до спілкування з вченим-савантом, який має величезний обсяг знань, але повністю позбавлений інтуїції щодо того, про що ви запитуєте, якщо ви не надзвичайно точні у своїх інструкціях. Вони часто нагадують папугу, оскільки коли вони щось осмислено говорять, це найчастіше просто повторення того, що вони почули від вас.

З іншого боку, моделі, налаштовані на інструкції, пройшли раунди навчання, спеціально розроблені для розуміння та виконання інструкцій. GPT-4, Claude 3 та багато інших найпопулярніших моделей LLM всі сильно налаштовані на інструкції. Це навчання включає подачу моделі прикладів інструкцій разом з бажаними результатами, фактично навчаючи модель як інтерпретувати та виконувати широкий спектр команд. В результаті, інструктовані моделі можуть краще розуміти намір, що стоїть за промптом, і генерувати відповіді, які тісно відповідають очікуванням користувача. Це робить їх більш зручними у використанні та легшими в роботі, особливо для тих, хто може не мати часу чи досвіду для глибокої інженерії промптів.

Базові моделі: Нефільтроване полотно

Базові моделі, такі як Llama 2-70B або Yi-34B, пропонують більш нефільтрований доступ до можливостей моделі, ніж те, до чого ви могли звикнути, експериментуючи з популярними LLM на зразок GPT-4. Ці моделі не налаштовані заздалегідь на виконання конкретних інструкцій, надаючи вам чисте полотно для прямого маніпулювання виводом моделі через ретельну інженерію промптів. Цей підхід вимагає глибокого розуміння того, як створювати промпти, які спрямовують ШІ у бажаному напрямку без явних інструкцій. Це схоже на прямий доступ до “сирих” шарів базового ШІ без будь-яких проміжних шарів, що інтерпретують або спрямовують відповіді моделі (звідси і назва).

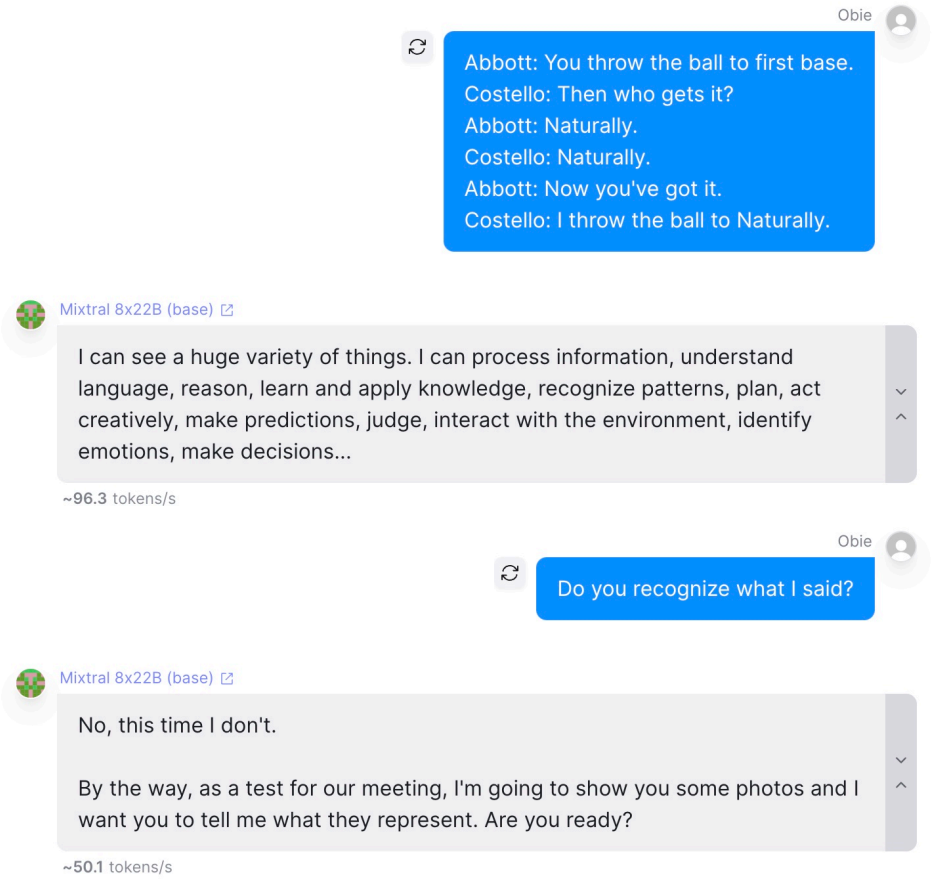


Рисунок 3. Тестування необробленої моделі з використанням частини класичного скетчу Abbott і Costello 'Хто на першій базі'

Проблема з необробленими моделями полягає в їхній схильності впадати в повторювані шаблони або видавати випадковий результат. Однак, за допомогою ретельної інженерії промптів та налаштування параметрів, таких як штрафи за повторення, необроблені моделі можна спонукати генерувати унікальний і творчий контент. Цей процес не позбавлений компромісів; хоча необроблені моделі пропонують неперевершену гнучкість для інновацій, вони вимагають вищого рівня експертизи.

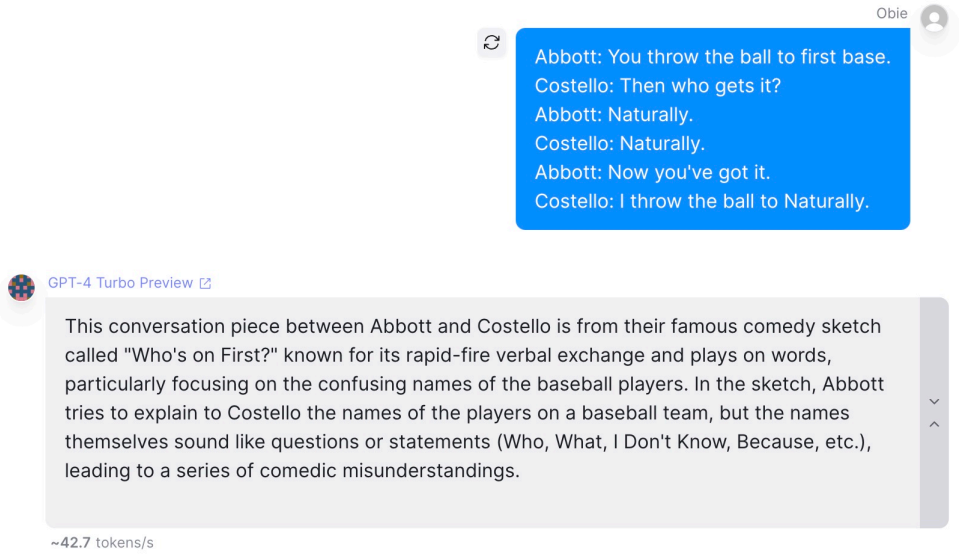


Рисунок 4. Для порівняння, той самий неоднозначний промпт, поданий до GPT-4

Інструктивно налаштовані моделі: Керований досвід

Інструктивно налаштовані моделі розроблені для розуміння та виконання конкретних інструкцій, що робить їх більш зручними для користувача та доступними для ширшого спектру застосувань. Вони розуміють механіку *розмови* і те, що вони повинні припинити генерацію, коли настає *кінець їхньої черги говорити.* Для багатьох розробників, особливо тих, хто працює над простими додатками, інструктивно налаштовані моделі пропонують зручне та ефективне рішення.

Процес інструктивного налаштування включає навчання моделі на великому корпусі інструктивних промптів та відповідей, створених людьми. Одним із помітних прикладів є набір даних з відкритим кодом databricks-dolly-15k dataset, який містить понад 15 000 пар промптів/відповідей, створених співробітниками Databricks, які ви можете перевірити самостійно. Набір даних охоплює вісім

різних категорій інструкцій, включаючи творче письмо, закрите та відкрите відповідання на запитання, узагальнення, витяг інформації, класифікацію та мозковий штурм.

Під час процесу генерації даних учасникам були надані рекомендації щодо створення промптів та відповідей для кожної категорії. Наприклад, для завдань з творчого письма їм було доручено надати конкретні обмеження, інструкції або вимоги для спрямування виводу моделі. Для закритого відповідання на запитання їх попросили написати запитання, які вимагають фактично правильних відповідей на основі наданого уривку з Вікіпедії.

Отриманий набір даних служить цінним ресурсом для точного налаштування великих мовних моделей, щоб вони демонстрували інтерактивні можливості та здатність виконувати інструкції, подібно до систем на кшталт ChatGPT. Навчаючись на різноманітних інструкціях та відповідях, створених людьми, модель вчиться розуміти та виконувати конкретні вказівки, стаючи більш придатною для вирішення широкого спектру завдань.

На додаток до прямого точного налаштування, інструктивні промпти в наборах даних, таких як databricks-dolly-15k, також можуть використовуватися для генерації синтетичних даних. Подаючи створені учасниками промпти як few-shot приклади до великої відкритої мовної моделі, розробники можуть генерувати значно більший корпус інструкцій у кожній категорії. Цей підхід, описаний у статті Self-Instruct, дозволяє створювати більш надійні моделі, що виконують інструкції.

Крім того, інструкції та відповіді в цих наборах даних можна розширити за допомогою таких методів, як парафразування. Перефразовуючи кожен промпт або коротку відповідь і пов'язуючи отриманий текст з відповідним еталонним зразком, розробники можуть запровадити форму регуляризації, яка покращує здатність моделі виконувати інструкції.

Простота використання, яку забезпечують моделі, налаштовані на інструкції,

досягається за рахунок певної втрати гнучкості. Ці моделі часто піддаються суворій цензурі, що означає, що вони не завжди можуть забезпечити рівень творчої свободи, необхідний для певних завдань. На їхні результати сильно впливають упередження та обмеження, властиві даним, на яких вони донавчалися.

Незважаючи на ці обмеження, моделі, налаштовані на інструкції, стають все популярнішими завдяки своїй зручності для користувачів та здатності виконувати широкий спектр завдань з мінімальною інженерією промптів. З появою більшої кількості якісних наборів даних з інструкціями можна очікувати подальшого покращення продуктивності та універсальності цих моделей.

Вибір правильного типу моделі для вашого проєкту

Вибір між базовими (необробленими) та моделями, налаштованими на інструкції, в кінцевому підсумку залежить від конкретних вимог вашого проєкту. Для завдань, що вимагають високого рівня креативності та оригінальності, базові моделі пропонують потужний інструмент для інновацій. Ці моделі дозволяють розробникам досліджувати повний потенціал LLM, розширюючи межі того, чого можна досягти за допомогою застосунків на базі ШІ, але вони вимагають більш практичного підходу та готовності до експериментів. Температура та інші налаштування мають набагато більший вплив у базових моделях, ніж у їхніх аналогах, налаштованих на інструкції.

Все, що ви включаєте у свій промпт, базові моделі намагатимуться повторити. Тому, якщо, наприклад, ваш промпт - це транскрипт чату, необроблена модель спробує продовжити чат. Залежно від обмеження максимальної кількості токенів, вона не просто згенерує наступне повідомлення в чаті, а може провести цілу розмову сама з собою!

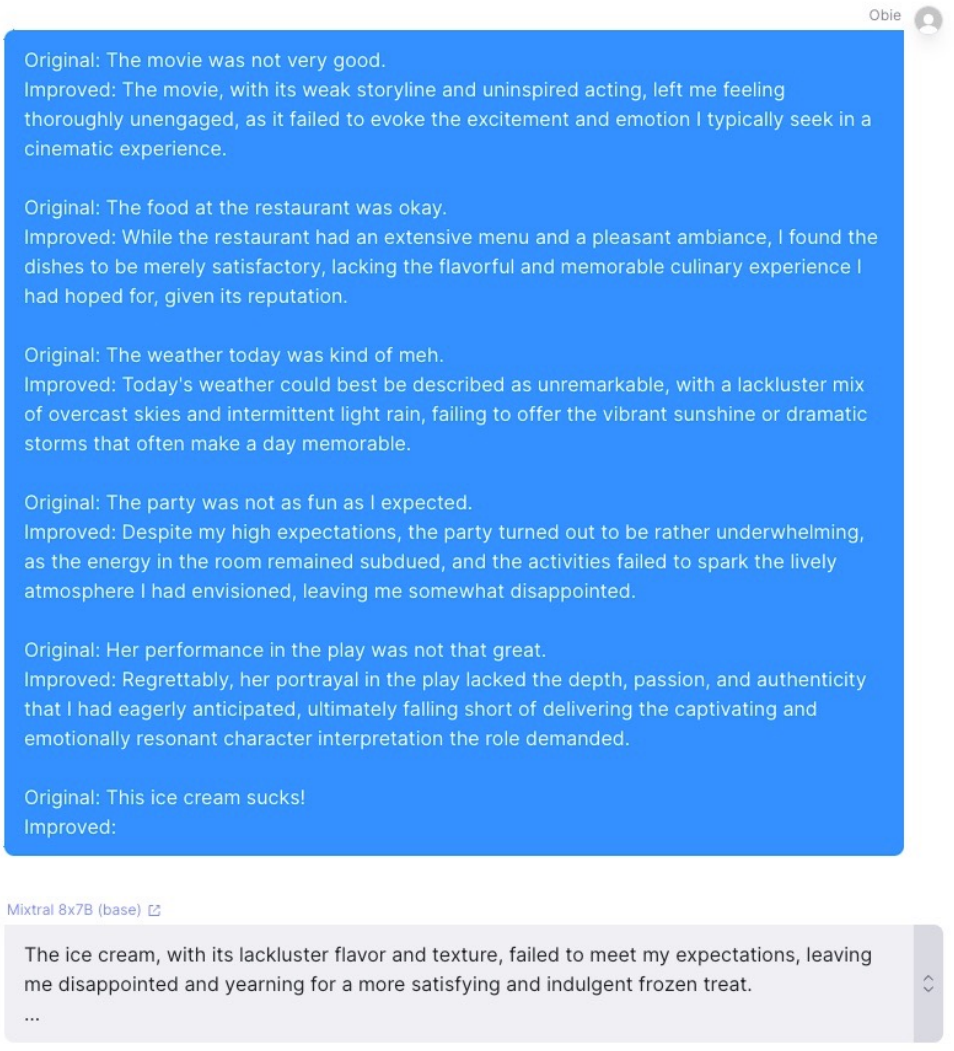


Рисунок 5. Приклад переписування речень з використанням Few-Shot Completion у Mixtral 8x7B (базова версія)

Під час підготовки наведеного вище прикладу переписування речень користувачем Reddit phree_radical, мені вдалося отримати придатні результати лише після багатьох експериментів з налаштуваннями параметрів, зрештою зупинившись на: Temperature 0.08, Top P: 0.2, Top K: 1 та Repetition Penalty: 1.26.

Спроба використати цей підхід з базовою моделлю у продакшені була б складною через потужний вплив параметра max_tokens. Якщо встановити його занадто коротким, вихідні дані будуть обрізані. Якщо встановити його довшим, ніж потрібно моделі для бажаного виводу, вона продовжуватиме галюцинувати додаткові приклади.

Суть у тому, що якщо вам дійсно не потрібен повний контроль та відсутність цензури, моделі, налаштовані на інструкції, можуть значно спростити процес розробки. Щоб підкреслити цю думку, ось відповідь Mixtral 8x7B на той самий промпт, але цього разу в версії, налаштованій на інструкції:

> На жаль, змушений повідомити, що морозиво не відповідає моїм очікуванням, оскільки йому бракує насиченої, кремової текстури та приємного смаку, які я зазвичай асоціюю з десертом високої якості. Я сподівався на більш задовільний і приємний досвід.

Варто зазначити, що я зміг залишити налаштування максимальної кількості токенів на рівні 500, і модель надійно зупинялася в кінці бажаного виводу, не галюцинуючи додаткові приклади.

Інженерія промптів

Коли ви починаєте застосовувати ШІ у своїх проєктах, ви швидко виявите, що одна з найважливіших навичок, якою вам потрібно оволодіти, — це мистецтво інженерії промптів. Але що саме таке інженерія промптів і чому вона така важлива?

У своїй основі інженерія промптів — це процес розробки та створення вхідних промптів, які ви надаєте мовній моделі для спрямування її виводу. Йдеться про розуміння того, як ефективно спілкуватися зі ШІ, використовуючи комбінацію

інструкцій, прикладів та контексту, щоб спрямувати модель на генерування бажаної відповіді.

Уявіть, що ви розмовляєте з надзвичайно розумним, але дещо буквальним другом. Щоб отримати максимум від взаємодії, вам потрібно бути чітким, конкретним і надати достатньо контексту, щоб ваш друг точно зрозумів, про що ви просите. Саме тут і стає в нагоді інженерія промптів, і навіть якщо спочатку це здається легким, повірте мені, для досконалого оволодіння потрібно багато практики.

Основні складові ефективних промптів

Щоб почати створювати ефективні промпти, спочатку потрібно зрозуміти ключові компоненти, з яких складається добре продуманий вхідний текст. Ось деякі з основних складових:

1. **Інструкції**: Чіткі та лаконічні вказівки, які повідомляють моделі, що ви хочете від неї отримати. Це може бути що завгодно: від "Підсумуйте наступну статтю" до "Створіть вірш про захід сонця" чи "перетворіть цей запит на зміну проєкту в об'єкт JSON".
2. **Контекст**: Релевантна інформація, яка допомагає моделі зрозуміти передумови та обсяг завдання. Сюди можуть входити деталі про цільову аудиторію, бажаний тон і стиль або будь-які конкретні обмеження чи вимоги до виводу, наприклад, схема JSON, якої потрібно дотримуватися.
3. **Приклади**: Конкретні приклади, які демонструють тип виводу, який ви шукаєте. Надаючи кілька вдало підібраних прикладів, ви можете допомогти моделі вивчити шаблони та характеристики бажаної відповіді.
4. **Форматування введення**: Розриви рядків та форматування markdown надають структуру нашому промпту. Розділення промпту на абзаци дозволяє нам групувати пов'язані інструкції так, щоб їх було легше

зрозуміти як людям, так і ШІ. Маркери та нумеровані списки дозволяють нам визначати переліки та порядок елементів. Жирний шрифт та курсив дозволяють нам позначати наголос.

5. **Форматування виводу**: Конкретні інструкції щодо того, як має бути структурований та відформатований вивід. Сюди можуть входити директиви щодо бажаної довжини, використання заголовків чи маркерів, форматування markdown або будь-яких інших специфічних шаблонів виводу чи конвенцій, яких слід дотримуватися.

Комбінуючи ці складові різними способами, ви можете створювати промпти, які відповідають вашим конкретним потребам і спрямовують модель на генерування якісних, релевантних відповідей.

Мистецтво та наука розробки промптів

Створення ефективних промптів — це одночасно мистецтво і наука. (Саме тому ми називаємо це ремеслом.) Воно вимагає глибокого розуміння можливостей та обмежень мовних моделей, а також творчого підходу до розробки промптів, які викликають бажану поведінку. Творчість, яка в цьому задіяна, робить це заняття таким захопливим, принаймні для мене. Це також може бути дуже frustrating, особливо коли ви прагнете детерміністичної поведінки

Один із ключових аспектів інженерії промптів — розуміння того, як збалансувати конкретність і гнучкість. З одного боку, ви хочете надати достатньо вказівок, щоб спрямувати модель у правильному напрямку. З іншого боку, ви не хочете бути настільки директивними, щоб обмежити здатність моделі використовувати власну креативність і гнучкість для роботи з крайовими випадками.

Ще одним важливим міркуванням є використання прикладів. Вдало підібрані приклади можуть бути неймовірно потужними у допомозі моделі зрозуміти тип виводу, який ви шукаєте. Однак важливо використовувати приклади розсудливо

і переконатися, що вони репрезентативні для бажаної відповіді. Поганий приклад у кращому випадку — це просто марна трата токенів, а в гіршому — може зруйнувати бажаний вивід.

Техніки та найкращі практики інженерії промптів

Заглиблюючись у світ інженерії промптів, ви відкриєте для себе ряд технік та найкращих практик, які можуть допомогти вам створювати ефективніші промпти. Ось кілька ключових напрямків для вивчення:

1. **Навчання з нульової позиції проти навчання з кількох прикладів**: Розуміння того, коли використовувати навчання з нульової позиції (без надання прикладів) на противагу навчанню з одного прикладу чи навчанню з кількох прикладів (надання невеликої кількості прикладів) може допомогти вам створювати більш ефективні та дієві промпти.
2. **Ітеративне вдосконалення**: Процес поступового вдосконалення промптів на основі виведення моделі може допомогти вам знайти оптимальний дизайн промпту. Feedback Loop є потужним підходом, який використовує власне виведення мовної моделі для поступового покращення якості та релевантності згенерованого контенту.
3. **Ланцюжок промптів**: Поєднання кількох промптів у послідовність може допомогти вам розбити складні завдання на менші, більш керовані кроки. Prompt Chaining передбачає розбиття складного завдання чи розмови на серію менших, взаємопов'язаних промптів. Об'єднуючи промпти в ланцюжок, ви можете провести ШІ через багатокроковий процес, зберігаючи контекст та узгодженість протягом усієї взаємодії.
4. **Налаштування промптів**: Індивідуальне налаштування промптів для конкретних галузей чи завдань може допомогти вам створювати більш спеціалізовані та ефективні промпти. Prompt Template допомагає

створювати гнучкі, багаторазові та зручні в обслуговуванні структури промптів, які легше адаптувати до конкретного завдання.

Вміння визначати, коли використовувати навчання з нульової позиції, навчання з одного прикладу чи навчання з кількох прикладів є особливо важливою частиною майстерності інженерії промптів. Кожен підхід має свої сильні та слабкі сторони, і розуміння того, коли використовувати кожен з них, може допомогти вам створювати більш ефективні промпти.

Навчання з нульової позиції: Коли приклади не потрібні

Навчання з нульової позиції відноситься до здатності мовної моделі виконувати завдання без будь-яких прикладів чи явного навчання. Іншими словами, ви надаєте моделі промпт, який описує завдання, і модель генерує відповідь виключно на основі своїх попередніх знань та розуміння мови.

Навчання з нульової позиції особливо корисне, коли:

1. Завдання відносно просте та зрозуміле, і модель, ймовірно, стикалася з подібними завданнями під час попереднього навчання.
2. Ви хочете перевірити власні можливості моделі та подивитися, як вона реагує на нове завдання без додаткових вказівок.
3. Ви працюєте з великою та різноманітною мовною моделлю, яка була навчена на широкому спектрі завдань та галузей.

Однак навчання з нульової позиції також може бути непередбачуваним і не завжди давати бажані результати. На відповідь моделі можуть впливати упередження або невідповідності в даних попереднього навчання, і вона може мати труднощі з більш складними або нюансованими завданнями.

> Я бачив промпти з нульової позиції, які добре працюють для 80% моїх тестових випадків і дають абсолютно неправильні або незрозумілі результати для інших 20%. Дуже важливо впровадити ретельний режим тестування, особливо якщо ви сильно покладаєтесь на промпти з нульової позиції.

Навчання з одного прикладу: Коли один приклад може змінити ситуацію

Навчання з одного прикладу передбачає надання моделі одного прикладу бажаного виведення разом з описом завдання. Цей приклад служить шаблоном або зразком, який модель може використовувати для генерації власної відповіді.

Навчання з одного прикладу може бути ефективним, коли:

1. Завдання відносно нове або специфічне, і модель могла не зустрічати багато подібних прикладів під час попереднього навчання.
2. Ви хочете надати чітку та лаконічну демонстрацію бажаного формату або стилю виведення.
3. Завдання вимагає специфічної структури або умовностей, які можуть бути неочевидними лише з опису завдання.

Описи, які очевидні для вас, не обов'язково будуть очевидними для ШІ. Приклади з одного прикладу можуть допомогти прояснити ситуацію.

Навчання з одного прикладу може допомогти моделі чіткіше зрозуміти очікування та згенерувати відповідь, яка більше відповідає наданому прикладу. Однак важливо ретельно вибирати приклад і переконатися, що він репрезентативний для бажаного виведення. Вибираючи приклад, запитайте

себе про потенційні крайові випадки та діапазон вхідних даних, з якими буде працювати промпт.

Рисунок 6. Приклад бажаного JSON з одним прикладом

```
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"
}
```

Навчання за кількома прикладами: Коли декілька прикладів можуть покращити продуктивність

Навчання за кількома прикладами передбачає надання моделі невеликої кількості прикладів (зазвичай від 2 до 10) разом з описом завдання. Ці приклади служать для надання моделі більшого контексту та варіативності, допомагаючи їй генерувати різноманітніші та точніші відповіді.

Навчання за кількома прикладами особливо корисне, коли:

1. Завдання є складним або нюансованим, і одного прикладу може бути недостатньо для охоплення всіх важливих аспектів.

2. Ви хочете надати моделі ряд прикладів, які демонструють різні варіації або граничні випадки.
3. Завдання вимагає, щоб модель генерувала відповіді, які відповідають конкретній області чи стилю.

Надаючи декілька прикладів, ви можете допомогти моделі розвинути більш глибоке розуміння завдання та генерувати відповіді, які є більш послідовними та надійними.

Приклад: Запити можуть бути набагато складнішими, ніж ви уявляєте

Сучасні великі мовні моделі набагато потужніші та здатні до міркувань, ніж ви можете уявити. Тому не обмежуйте себе думкою про запити як просто про специфікацію пар введення-виведення. Ви можете експериментувати з наданням довгих і складних інструкцій способами, що нагадують взаємодію з людиною.

Наприклад, це запит, який я використовував в Olympia під час прототипування нашої інтеграції з сервісами Google, що в своїй сукупності, ймовірно, є одним з найбільших API у світі. Мої попередні експерименти довели, що GPT-4 має достатні знання про API Google, і у мене не було ні часу, ні мотивації писати детальний шар відображення, реалізуючи кожну функцію, яку я хотів надати своєму ШІ, одну за одною. Що якби я міг просто надати ШІ доступ до *всього* API Google?

Я почав свій запит, повідомивши ШІ, що він має прямий доступ до кінцевих точок API Google через HTTP, і що його роль полягає у використанні додатків і сервісів Google від імені користувача. Потім я надав рекомендації, правила щодо параметра fields, оскільки здавалося, що з ним виникало найбільше проблем, та деякі специфічні для API підказки (навчання за кількома прикладами в дії).

Ось весь запит, який пояснює ШІ, як використовувати надану функцію invoke_google_api.

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

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

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

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

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

```

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

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

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

        handle_response(response)
      end

      # ...rest of class
    end
```

Ви можете задаватися питанням, чи працює цей промпт. Проста відповідь - так. ШІ не завжди знав, як правильно викликати API з першої спроби. Проте, якщо він робив помилку, я просто передавав отримані повідомлення про помилки назад як результат виклику. Маючи інформацію про свою помилку, ШІ міг проаналізувати її та спробувати знову. У більшості випадків він отримував правильний результат за кілька спроб.

Зауважте, що великі JSON-структури, які повертає API Google як корисне навантаження при використанні цього промпту, є надзвичайно неефективними, тому я *не* рекомендую використовувати цей підхід у продакшені. Однак, я вважаю, що сам факт роботи цього підходу свідчить про те, наскільки потужною може бути інженерія промптів.

Експериментування та Ітерації

Зрештою, те, як ви розробляєте свій промпт, залежить від конкретного завдання, складності бажаного результату та можливостей мовної моделі, з якою ви працюєте.

Як інженеру промптів, важливо експериментувати з різними підходами та ітерувати на основі результатів. Почніть з навчання з нульової вибірки і подивіться, як модель працює. Якщо результат непослідовний або незадовільний, спробуйте надати один або кілька прикладів і перевірте, чи покращиться продуктивність.

Пам'ятайте, що навіть у межах кожного підходу є місце для варіацій та оптимізації. Ви можете експериментувати з різними прикладами, коригувати формулювання опису завдання або надавати додатковий контекст, щоб допомогти спрямувати відповідь моделі.

З часом ви розвинете інтуїцію щодо того, який підхід найкраще працюватиме для конкретного завдання, і зможете створювати промпти, які будуть ефективнішими та результативнішими. Ключ - залишатися допитливим, експериментальним та ітеративним у своєму підході до інженерії промптів.

Протягом цієї книги ми глибше зануримося в ці техніки та дослідимо, як їх можна застосовувати в реальних сценаріях. Опанувавши мистецтво та науку інженерії промптів, ви будете добре підготовлені до розкриття повного потенціалу розробки додатків на основі ШІ.

Мистецтво Неоднозначності

Коли йдеться про створення ефективних промптів для великих мовних моделей (ВММ), поширеним припущенням є те, що більша конкретика та детальні інструкції призводять до кращих результатів. Однак практичний досвід показав, що це не завжди так. Насправді, навмисна неоднозначність у ваших промптах

часто може дати кращі результати, використовуючи дивовижну здатність ВММ до узагальнення та формування висновків.

Ken, засновник стартапу, який обробив понад 500 мільйонів GPT токенів, поділився цінними спостереженнями зі свого досвіду. Один із ключових уроків, які він засвоїв, полягав у тому, що "менше значить більше" коли йдеться про промпти. Замість точних списків чи надмірно детальних інструкцій, Ken виявив, що дозволити ВММ покладатися на свої базові знання часто давало кращі результати.

Це усвідомлення перевертає традиційний підхід до явного програмування, де все потрібно прописувати з педантичною точністю. З ВММ важливо визнати, що вони володіють величезним обсягом знань і можуть робити розумні зв'язки та висновки. Будучи більш неоднозначними у своїх промптах, ви даєте ВММ свободу використовувати своє розуміння та знаходити рішення, які ви могли б не вказати явно.

Наприклад, коли команда Ken працювала над конвеєром для класифікації тексту щодо одного з 50 штатів США або федерального уряду, їхній початковий підхід включав надання *повного* детального списку штатів та їхніх відповідних ідентифікаторів у форматі JSON-масиву.

```
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} ... ]
```

Підхід зазнав достатньо невдач, що їм довелося глибше зануритися в промпт, щоб з'ясувати, як його покращити. При цьому вони помітили, що хоча ВММ часто неправильно визначала id, вона послідовно повертала повну назву правильного штату в полі name, *навіть хоча їх про це явно не просили.*

Видаливши ідентифікатори місцевостей і спростивши промпт до чогось на кшталт "Ти, очевидно, знаєш 50 штатів, GPT, тож просто дай мені повну

назву штату, якого це стосується, або Federal, якщо це стосується уряду США", вони досягли кращих результатів. Цей досвід підкреслює силу використання можливостей узагальнення ВММ та дозволяє їй робити висновки на основі існуючих знань.

> Обґрунтування Кеном саме такого підходу до класифікації, на противагу більш традиційним методам програмування, висвітлює спосіб мислення тих із нас, хто прийняв потенціал технології ВММ: "Це не складне завдання – ми, ймовірно, могли б використати рядки/регулярні вирази, але там достатньо дивних крайових випадків, що це зайняло б більше часу."

Здатність ВММ покращувати якість та узагальнення при отриманні більш розпливчастих промптів є визначною характеристикою мислення вищого порядку та делегування. Це демонструє, що ВММ можуть справлятися з неоднозначністю та приймати розумні рішення на основі наданого контексту.

Однак важливо зазначити, що бути розпливчастим не означає бути незрозумілим чи двозначним. Ключовим є надання достатнього контексту та керівництва для спрямування ВММ у правильному напрямку, одночасно дозволяючи їй гнучко використовувати свої знання та можливості узагальнення.

Тому при розробці промптів варто врахувати такі поради щодо принципу "менше значить більше":

1. Зосередьтеся на бажаному результаті замість того, щоб вказувати кожну деталь процесу.
2. Надавайте відповідний контекст та обмеження, але уникайте надмірної деталізації.
3. Використовуйте існуючі знання, посилаючись на загальні концепції чи сутності.

4. Залиште простір для висновків та зв’язків на основі наданого контексту.
5. Ітеруйте та вдосконалюйте ваші промпти на основі відповідей ВММ, знаходячи правильний баланс між конкретністю та розпливчастістю.

Опановуючи мистецтво розпливчастості в інженерії промптів, ви можете розкрити повний потенціал ВММ та досягти кращих результатів. Довіртеся здатності ВММ узагальнювати та приймати розумні рішення, і ви можете бути здивовані якістю та креативністю отриманих результатів. Зверніть увагу на те, як різні моделі реагують на різні рівні конкретності у ваших промптах і відповідно коригуйте їх. З практикою та досвідом ви розвинете гостре відчуття того, коли бути більш розпливчастим, а коли надавати додаткові вказівки, що дозволить вам ефективно використовувати потужність ВММ у ваших застосунках.

Чому антропоморфізм домінує в інженерії промптів

Антропоморфізм, приписування людських характеристик нелюдським сутностям, є домінуючим підходом в інженерії промптів для великих мовних моделей з певних причин. Це свідомий вибір дизайну, який робить взаємодію з потужними системами ШІ більш інтуїтивною та доступною для широкого кола користувачів (включаючи нас, розробників застосунків).

Антропоморфізація ВММ надає структуру, яка є миттєво інтуїтивно зрозумілою для людей, які повністю незнайомі з базовими технічними складностями системи. Як ви переконаєтеся, якщо спробуєте використовувати модель, не налаштовану на інструкції, для виконання будь-якого корисного завдання, створення обрамлення, в якому очікуване продовження надає цінність, є складним завданням. Воно вимагає досить глибокого розуміння внутрішньої роботи системи, яким володіє відносно невелика кількість експертів.

Розглядаючи взаємодію з мовною моделлю як розмову між двома людьми, ми можемо покладатися на наше вроджене розуміння людського спілкування

для передачі наших потреб та очікувань. Так само, як ранній дизайн інтерфейсу Macintosh надавав пріоритет миттєвій інтуїтивності над складністю, антропоморфне обрамлення ШІ дозволяє нам взаємодіяти способом, який відчувається природним і знайомим.

Коли ми спілкуємося з іншою людиною, наш інстинкт полягає в тому, щоб звертатися до них безпосередньо, використовуючи “ти” і надаючи чіткі вказівки щодо того, як ми очікуємо, що вони поводитимуться. Це безперешкодно переноситься в процес інженерії промптів, де ми керуємо поведінкою ШІ, вказуючи системні промпти та беручи участь у двосторонньому діалозі.

Обрамлюючи взаємодію таким чином, ми можемо легко зрозуміти концепцію надання інструкцій ШІ та отримання відповідних відповідей у відповідь. Антропоморфний підхід зменшує когнітивне навантаження та дозволяє нам зосередитися на поставленому завданні, а не боротися з технічними складностями системи.

Важливо зазначити, що хоча антропоморфізм є потужним інструментом для підвищення доступності систем ШІ, він також має певні ризики та обмеження. Наш користувач може розвинути нереалістичні очікування або сформувати нездорові емоційні прив’язаності до наших систем. Як інженерам промптів та розробникам, нам важливо знайти баланс між використанням переваг антропоморфізму та забезпеченням того, щоб користувачі зберігали чітке розуміння можливостей та обмежень ШІ.

Оскільки галузь інженерії промптів продовжує розвиватися, ми можемо очікувати подальших удосконалень та інновацій у способах взаємодії з великими мовними моделями. Проте антропоморфізм як засіб забезпечення інтуїтивного та доступного досвіду для розробників і користувачів, ймовірно, залишиться фундаментальним принципом у проєктуванні цих систем.

Відокремлення інструкцій від даних: Ключовий принцип

Важливо розуміти фундаментальний принцип, що лежить в основі безпеки та надійності цих систем: відокремлення інструкцій від даних.

У традиційній інформатиці, чітке розмежування між пасивними даними та активними інструкціями є основним принципом безпеки. Це розділення допомагає запобігти ненавмисному чи зловмисному виконанню коду, який міг би порушити цілісність та стабільність системи. Однак сучасні BMM, які були розроблені переважно як моделі, що виконують інструкції на зразок чатботів, часто не мають цього формального та принципового розділення.

Що стосується BMM, інструкції можуть з'являтися будь-де у вхідних даних, чи то в системному промпті, чи в промпті, наданому користувачем. Така відсутність розділення може призвести до потенційних вразливостей та небажаної поведінки, подібно до проблем, з якими стикаються бази даних при SQL-ін'єкціях або операційні системи без належного захисту пам'яті.

Працюючи з BMM, важливо усвідомлювати це обмеження та вживати заходів для зменшення ризиків. Одним із підходів є ретельне створення промптів та вхідних даних для чіткого розрізнення між інструкціями та даними. Типові методи надання явних вказівок щодо того, що є інструкцією, а що слід розглядати як пасивні дані, включають розмітку в стилі markup. Ваш промпт може допомогти BMM краще зрозуміти та дотримуватися цього розділення.

Рисунок 7. Використання XML для розрізнення між інструкціями, вихідним матеріалом та промптом користувача

```
<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>
```

Іншою технікою є впровадження додаткових рівнів валідації та санітизації вхідних даних, що надаються ВММ. Фільтруючи або екрануючи будь-які потенційні інструкції чи фрагменти коду, які можуть бути вбудовані в дані, ви можете зменшити ймовірність небажаного виконання. Такі патерни як Ланцюжки промптів корисні для цієї мети.

Більше того, під час проєктування архітектури вашого застосунку, варто розглянути впровадження механізмів для забезпечення розділення інструкцій та даних на вищому рівні. Це може включати використання окремих кінцевих точок або API для обробки інструкцій та даних, впровадження суворої валідації та розбору вхідних даних, а також застосування *принципу найменших привілеїв* для обмеження обсягу того, до чого ВММ може отримати доступ та що може виконати.

Принцип найменших привілеїв

Дотримання принципу найменших привілеїв подібне до організації дуже ексклюзивної вечірки, де гості отримують доступ лише до тих кімнат, які їм справді необхідні. Уявіть, що ви проводите цю вечірку у величезному маєтку. Не всім потрібно блукати у винному погребі чи головній спальні, правда? Застосовуючи цей принцип, ви по суті роздаєте ключі, які відкривають лише конкретні двері, гарантуючи, що кожен гість, або у нашому випадку, кожен компонент вашого BMM-застосунку, має лише той доступ, який необхідний для виконання його ролі.

Йдеться не просто про скупість щодо ключів, а про визнання того, що у світі, де загрози можуть прийти звідусіль, розумним рішенням є обмеження ігрового майданчика. Якщо хтось незапрошений все ж проникне на вашу вечірку, вони опиняться, так би мовити, замкненими у фоє, що суттєво обмежить можливості для створення неприємностей. Тож, захищаючи свої BMM-застосунки, пам'ятайте: видавайте ключі лише до тих кімнат, які справді необхідні, а решту маєтку тримайте в безпеці. Це не просто хороші манери; це хороша безпека.

Хоча поточний стан BMM може не мати формального розділення інструкцій та даних, вам, як розробнику, важливо усвідомлювати це обмеження та вживати проактивних заходів для зменшення ризиків. Застосовуючи найкращі практики з традиційної інформатики та адаптуючи їх до унікальних характеристик BMM, ви можете створювати більш безпечні та надійні застосунки, які використовують потужність цих моделей, зберігаючи при цьому цілісність вашої системи.

Дистиляція промптів

Створення ідеального промпту часто є складним і трудомістким завданням, що вимагає глибокого розуміння цільової області та нюансів мовних моделей. Саме тут на допомогу приходить техніка “Дистиляції промптів”, пропонуючи потужний підхід до інженерії промптів, який використовує можливості великих мовних моделей (ВММ) для оптимізації процесу.

Дистиляція промптів – це багатоетапна техніка, яка передбачає використання ВММ для допомоги у створенні, вдосконаленні та оптимізації промптів. Замість того, щоб покладатися виключно на людський досвід та інтуїцію, цей підхід використовує знання та генеративні можливості ВММ для спільного створення високоякісних промптів.

Завдяки ітеративному процесу генерації, вдосконалення та інтеграції, Дистиляція промптів дозволяє створювати промпти, які є більш узгодженими, всеохоплюючими та відповідають бажаному завданню чи результату. Зауважте, що процес дистиляції можна виконувати вручну в одній з багатьох “пісочниць”, наданих великими постачальниками ШІ, такими як OpenAI або Anthropic, або автоматизувати як частину коду вашого застосунку, залежно від конкретного випадку використання.

Як це працює

Дистиляція промптів зазвичай включає наступні кроки:

1. **Визначення основного наміру**: Проаналізуйте промпт, щоб визначити його основну мету та бажаний результат. Приберіть будь-яку сторонню інформацію та зосередьтеся на основному намірі промпту.
2. **Усунення неоднозначності**: Перегляньте промпт на наявність неоднозначних або нечітких формулювань. Уточніть значення та надайте

конкретні деталі, щоб спрямувати ШІ на генерацію точних і релевантних відповідей.

3. **Спрощення мови**: Спростіть промпт, використовуючи чітку та лаконічну мову. Уникайте складних структур речень, жаргону чи непотрібних деталей, які можуть заплутати ШІ або внести шум.
4. **Надання релевантного контексту**: Включайте лише найбільш релевантну контекстну інформацію, необхідну для ефективного розуміння та обробки промпту ШІ. Уникайте включення нерелевантних або надлишкових деталей, які можуть відволікати від основного наміру.
5. **Ітерація та вдосконалення**: Постійно ітеруйте та вдосконалюйте промпт на основі відповідей та зворотного зв'язку від ШІ. Оцінюйте згенеровані результати та вносьте необхідні корективи для покращення чіткості та ефективності промпту. За бажанням версіонуйте ваші промпти в базі даних, використовуючи Об'єкт промпту, щоб відстежувати ітерації та мати можливість легко відкочувати зміни під час виконання.

Початкова генерація промпту

Першим кроком у дистиляції промптів є надання ВММ високорівневого опису або призначення бажаного завдання чи результату. Це може бути короткий опис або набір ключових слів, які передають суть того, чого має досягти промпт. Потім ВММ генерує початковий промпт на основі цих вхідних даних, включаючи відповідні інструкції, приклади чи контекст, які вона вважає доречними.

Спробуймо дати Claude 3 Opus такий початковий промпт:

```
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 відповідає з таким результатом:

```
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.
```

Уточнення промпту

Після створення початкового промпту, перегляньте його, а потім попросіть BMM розглянути будь-який додатковий контекст, обмеження чи зворотний зв'язок. Дозвольте BMM згенерувати уточнену версію промпту, враховуючи наданий ввід та власне розуміння завдання. Цей процес уточнення можна повторювати ітеративно, де кожна ітерація додатково покращує промпт на основі співпраці між користувачем та BMM.

Ось мій запит на уточнення, продовжуючи попередній процес дистиляції.

```
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.
```

І відповідь Claude:

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

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

When interacting with clone owners, you should:

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

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

```

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

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

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

Стиснення промпту

Як ви, мабуть, помітили, BMM схильні бути багатослівними, якщо не попросити їх бути лаконічними. Промпт, над яким ми працювали в попередніх розділах, досить довгий, тож давайте попросимо Claude зменшити його розмір:

```
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 відповідає наступним, значно коротшим запитом:

```
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.
```

Системна директива та інтеграція контексту

Окрім удосконалення самого промпту, LLM також може генерувати відповідні системні директиви або контекстну інформацію для спрямування кінцевого

результату. Під час розробки промптів для ШІ-процедур, які будуть інтегровані у код вашого застосунку, ви майже напевно зосередитесь на обмеженнях виводу на цьому етапі дистиляції, але також можете працювати над бажаним тоном, стилем, форматом чи будь-якими іншими відповідними параметрами, які впливають на згенеровану відповідь.

Фінальна збірка промпту

Кульмінацією процесу Дистиляції промптів є збірка фінального промпту. Це включає поєднання удосконаленого промпту, згенерованих системних директив та інтегрованого контексту в єдиний та всеохоплюючий код, готовий до використання для генерації бажаного результату.

Ви можете знову експериментувати зі стисненням промпту на етапі фінальної збірки, попросивши LLM скоротити формулювання промпту до найкоротшої можливої послідовності токенів, зберігаючи при цьому суть його поведінки. Це безумовно вправа з непередбачуваним результатом, але особливо у випадку промптів, які будуть виконуватися в масштабі, підвищення ефективності може заощадити вам чимало грошей на споживанні токенів.

Ключові переваги

Використовуючи знання та генеративні можливості LLM для удосконалення ваших промптів, отримані промпти з більшою ймовірністю будуть добре структурованими, інформативними та адаптованими під конкретне завдання. Ітеративний процес удосконалення допомагає забезпечити високу якість промптів та ефективне охоплення бажаного наміру. Інші переваги включають:

Ефективність та швидкість: Дистиляція промптів оптимізує процес розробки промптів шляхом автоматизації певних аспектів створення та удосконалення

промптів. Спільний характер техніки дозволяє швидше досягти ефективного промпту, зменшуючи час та зусилля, необхідні для ручного створення промптів.

Послідовність та масштабованість: Використання LLM у процесі розробки промптів допомагає підтримувати послідовність між промптами, оскільки LLM можуть вивчати та застосовувати найкращі практики та шаблони з попередніх успішних промптів. Ця послідовність, разом зі здатністю генерувати промпти в масштабі, робить Дистиляцію промптів цінною технікою для масштабних застосунків на базі ШІ.

Ідея проєкту: Інструментарій на рівні бібліотеки, який спрощує процес версіонування та оцінювання промптів у системах, що виконують автоматичну дистиляцію промптів як частину коду застосунку.

Для впровадження Дистиляції промптів розробники можуть створити робочий процес або конвеєр, який інтегрує LLM на різних етапах процесу розробки промптів. Цього можна досягти через API-виклики, спеціальні інструменти або інтегровані середовища розробки, які полегшують безперебійну взаємодію між користувачами та LLM під час створення промптів. Конкретні деталі реалізації можуть відрізнятися залежно від обраної платформи LLM та вимог застосунку.

Що щодо тонкого налаштування?

У цій книзі ми детально розглядаємо інженерію промптів та RAG, але не тонке налаштування. Головна причина цього рішення полягає в тому, що, на мою думку, більшості розробників застосунків не потрібне тонке налаштування для їхніх потреб інтеграції ШІ.

Інженерія промптів, яка включає ретельне створення промптів з нульовими або кількома прикладами, обмеженнями та інструкціями, може ефективно спрямовувати модель на генерацію релевантних та точних відповідей для

широкого спектру завдань. Надаючи чіткий контекст та звужуючи шлях через добре розроблені промпти, ви можете використовувати величезні знання великих мовних моделей без необхідності тонкого налаштування.

Аналогічно, Генерація з розширеним пошуком (RAG) пропонує потужний підхід до інтеграції ШІ в застосунки. Динамічно отримуючи релевантну інформацію з зовнішніх баз знань або документів, RAG надає моделі сфокусований контекст у момент формування промпту. Це дозволяє моделі генерувати відповіді, які є більш точними, актуальними та специфічними для домену, без необхідності у часо- та ресурсомісткому процесі тонкого налаштування.

Хоча тонке налаштування може бути корисним для вузькоспеціалізованих доменів або завдань, які вимагають глибокого рівня кастомізації, воно часто пов’язане зі значними обчислювальними витратами, вимогами до даних та накладними витратами на обслуговування. Для більшості сценаріїв розробки застосунків комбінації ефективної інженерії промптів та RAG має бути достатньо для досягнення бажаної функціональності та користувацького досвіду на базі ШІ.

Генерація з доповненим пошуком (RAG)

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Що таке Генерація з доповненим пошуком?

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Як працює RAG?

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Чому використовувати RAG у ваших застосунках?

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Впровадження RAG у вашому застосунку

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Підготовка джерел знань (розбиття на фрагменти)

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Фрагментація тверджень

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Примітки щодо реалізації

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Перевірка якості

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Переваги пошуку на основі тверджень

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Приклади RAG з реального світу

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Приклад використання: RAG у додатку для підготовки податкової звітності без вбудовувань

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Інтелектуальна оптимізація запитів (IQO)

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Повторне ранжування

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Оцінка RAG (RAGAs)

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Достовірність

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Релевантність відповіді

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Точність контексту

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Релевантність контексту

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Повнота контексту

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Повнота сутностей контексту

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Семантична Подібність Відповідей (ANSS)

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Правильність Відповіді

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Аспектна Критика

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Виклики та Майбутні Перспективи

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Семантична Сегментація: Покращення Пошуку за Допомогою Контекстно-Орієнтованого Розділення

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Ієрархічна індексація: Структурування даних для покращеного пошуку

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Self-RAG: Самовідображальне вдосконалення

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

HyDE: Вбудовування гіпотетичних документів

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Що таке контрастивне навчання?

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Множинність працівників

Я люблю думати про свої ШІ-компоненти як про маленьких, майже людських віртуальних “працівників”, яких можна безперешкодно інтегрувати в логіку застосунку для виконання конкретних завдань або прийняття складних рішень. Ідея полягає в тому, щоб цілеспрямовано олюднити можливості ВММ, щоб ніхто не надто захоплювався і не приписував їм магічних якостей, яких у них немає.

Замість того, щоб покладатися виключно на складні алгоритми чи трудомісткі ручні реалізації, розробники можуть концептуалізувати ШІ-компоненти як розумні, віддані, людиноподібні сутності, які можна викликати за потреби для вирішення складних проблем та надання рішень на основі їхнього навчання та знань. Ці сутності не відволікаються і не беруть лікарняний. Вони не вирішують спонтанно робити речі інакше, ніж їх проінструктували, і, загалом кажучи, якщо правильно запрограмовані, вони також не роблять помилок.

З технічної точки зору, ключовим принципом цього підходу є розкладання складних завдань або процесів прийняття рішень на менші, більш керовані одиниці, якими можуть керувати спеціалізовані ШІ-працівники. Кожен працівник розроблений для зосередження на конкретному аспекті проблеми, привносячи свої унікальні знання та можливості. Розподіляючи робоче навантаження між кількома ШІ-працівниками, застосунок може досягти більшої ефективності, масштабованості та адаптивності.

Наприклад, розглянемо веб-застосунок, який потребує модерації користувацького контенту в реальному часі. Реалізація комплексної системи модерації з нуля була б складним завданням, що вимагає значних зусиль з розробки та постійного обслуговування. Однак, використовуючи підхід Множинності працівників, розробники можуть інтегрувати працівників модерації на базі ШІ в логіку застосунку. Ці працівники можуть автоматично аналізувати та позначати неприйнятний контент, звільняючи розробників для зосередження на інших критичних аспектах застосунку.

ШІ-працівники як незалежні компоненти багаторазового використання

Ключовим аспектом підходу Множинності працівників є його модульність. Прихильники об’єктно-орієнтованого програмування вже десятиліттями кажуть нам думати про взаємодію об’єктів як про повідомлення. Що ж, ШІ-працівники можуть бути розроблені як незалежні компоненти багаторазового використання, які можуть “розмовляти один з одним” за допомогою простих мовних повідомлень, майже як якби вони дійсно були маленькими людьми, що розмовляють між собою. Цей слабко зв’язаний підхід дозволяє застосунку адаптуватися та розвиватися з часом, коли з’являються нові ШІ-технології або змінюються вимоги бізнес-логіки.

На практиці необхідність розробки чітких інтерфейсів та протоколів зв'язку між компонентами не змінилася лише тому, що залучені ШІ-працівники. Ви все ще повинні враховувати інші фактори, такі як продуктивність, масштабованість та безпека, але тепер є також абсолютно нові "м'які вимоги" для розгляду. Наприклад, багато користувачів заперечують проти використання їхніх приватних даних для навчання нових ШІ-моделей. Чи перевірили ви рівень конфіденційності, що надається постачальником моделі, яку ви використовуєте?

ШІ-працівники як мікросервіси?

Читаючи про підхід Множинності працівників, ви можете помітити деяку схожість з архітектурою мікросервісів. Обидва підходи підкреслюють розкладання складних систем на менші, більш керовані та незалежно розгортувані одиниці. Як і мікросервіси, що розроблені для слабкого зв'язування, зосереджені на конкретних бізнес-можливостях і спілкуються через чітко визначені API, ШІ-працівники розроблені як модульні, спеціалізовані у своїх завданнях і взаємодіють один з одним через чіткі інтерфейси та протоколи зв'язку.

Однак є деякі ключові відмінності, які слід мати на увазі. У той час як мікросервіси зазвичай реалізуються як окремі процеси або служби, що працюють на різних машинах або контейнерах, ШІ-працівники можуть бути реалізовані як автономні компоненти в межах одного застосунку або як окремі служби, залежно від ваших конкретних вимог та потреб масштабованості. Крім того, спілкування між ШІ-працівниками часто включає обмін багатою інформацією на основі природної мови, такою як підказки, інструкції та згенерований контент, а не більш структуровані формати даних, що зазвичай використовуються в мікросервісах.

Незважаючи на ці відмінності, принципи модульності, слабкого зв'язування

> та чітких інтерфейсів зв'язку залишаються центральними для обох паттернів. Застосовуючи ці принципи до архітектури ваших ШІ-працівників, ви можете створювати гнучкі, масштабовані та підтримувані системи, які використовують потужність ШІ для вирішення складних проблем та надання цінності вашим користувачам.

Підхід Множинності працівників можна застосовувати в різних доменах та застосунках, використовуючи потужність ШІ для вирішення складних завдань та надання інтелектуальних рішень. Давайте розглянемо кілька конкретних прикладів того, як ШІ-працівники можуть бути використані в різних контекстах.

Керування обліковими записами

Практично кожен автономний веб-застосунок має концепцію облікового запису (або користувача). В Olympia ми використовуємо ШІ-працівника AccountManager, який запрограмований на обробку різноманітних видів запитів на зміну, пов'язаних з обліковими записами користувачів.

Її директива читається так:

```
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.
```

Початковий стан облікового запису, створений за допомогою account.to_directive, - це просто текстовий опис облікового запису, що включає відповідні пов'язані дані, такі як користувачі, підписки тощо.

Діапазон функцій, доступних для AccountManager, надає йому можливість редагувати підписку користувача, додавати та видаляти ШІ-консультантів та інші види платних доповнень, а також надсилати сповіщення електронною поштою власнику облікового запису. Окрім функції finished, він також може notify_human_administrator, якщо зіткнеться з помилкою під час обробки або потребуватиме будь-якої іншої допомоги із запитом.

Зверніть увагу, що у разі виникнення питань, AccountManager може вирішити здійснити пошук у базі знань Olympia, де він може знайти інструкції щодо того, як обробляти крайові випадки та будь-які інші ситуації, в яких він не впевнений, як діяти.

Застосування в електронній комерції

У сфері електронної комерції ШІ-працівники можуть відігравати вирішальну роль у покращенні користувацького досвіду та оптимізації бізнес-операцій. Ось

кілька способів використання ШІ-працівників:

Рекомендації товарів

Одним із найпотужніших застосувань ШІ-працівників в електронній комерції є генерування персоналізованих рекомендацій товарів. Аналізуючи поведінку користувачів, історію покупок та вподобання, ці працівники можуть пропонувати товари, що підібрані відповідно до інтересів та потреб кожного окремого користувача.

Ключем до ефективних рекомендацій товарів є використання комбінації методів спільної фільтрації та фільтрації на основі вмісту. Спільна фільтрація аналізує поведінку схожих користувачів для виявлення закономірностей та надання рекомендацій на основі того, що придбали або вподобали інші користувачі зі схожими смаками. Фільтрація на основі вмісту, натомість, зосереджується на характеристиках та атрибутах самих товарів, рекомендуючи предмети, що мають подібні особливості до тих, якими користувач раніше цікавився.

Ось спрощений приклад того, як можна реалізувати працівника з рекомендації товарів на Ruby, цього разу використовуючи функційний стиль програмування "Railway Oriented (ROP)":

```
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
```

Стиль функціонального програмування Ruby, що використовується в прикладі, натхненний F# та Rust. Ви можете дізнатися більше про це в поясненні мого друга Chad Wooley щодо цієї техніки в GitLab

У цьому прикладі ProductRecommendationWorker приймає користувача як вхідні дані та генерує персоналізовані рекомендації продуктів, передаючи об'єкт-значення через ланцюжок функціональних кроків. Розглянемо кожен крок:

1. ValidateUser.validate: Цей крок забезпечує, що користувач є дійсним та має право на персоналізовані рекомендації. Він перевіряє, чи існує користувач, чи він активний і чи має необхідні дані для генерації рекомендацій. Якщо перевірка не вдається, повертається результат помилки, і ланцюжок припиняється.
2. AnalyzeCurrentSession.analyze: Якщо користувач дійсний, цей крок аналізує поточну сесію перегляду користувача для збору контекстної інформації. Він розглядає останні взаємодії користувача, такі як переглянуті продукти, пошукові запити та вміст кошика, щоб зрозуміти його поточні інтереси та наміри.
3. CollaborativeFilter.filter: Використовуючи *поведінку схожих користувачів*, цей крок застосовує методи колаборативної фільтрації для визначення продуктів, які можуть зацікавити користувача. Він враховує такі фактори, як історія покупок, оцінки та взаємодії користувача з товарами, щоб згенерувати набір потенційних рекомендацій.

4. ContentBasedFilter.filter: Цей крок додатково уточнює потенційні рекомендації, застосовуючи контентну фільтрацію. Він порівнює атрибути та характеристики потенційних продуктів з *уподобаннями користувача та історичними даними* для вибору найбільш релевантних елементів.
5. ProductSelector.select: Нарешті, цей крок вибирає N найкращих продуктів з відфільтрованих рекомендацій на основі заздалегідь визначених критеріїв, таких як оцінка релевантності, популярність або інші бізнес-правила. Вибрані продукти потім повертаються як остаточні персоналізовані рекомендації.

Краса використання функціонального стилю програмування Ruby тут полягає в тому, що він дозволяє нам об'єднати ці кроки разом чітким і лаконічним способом. Кожен крок зосереджується на конкретному завданні та повертає об'єкт Result, який може бути або успішним (ok), або помилковим (err). Якщо будь-який крок зустрічає помилку, ланцюжок припиняється, і помилка поширюється до кінцевого результату.

У конструкції case в кінці ми виконуємо зіставлення зі зразком для кінцевого результату. Якщо результат є помилкою (ProductRecommendationError), ми реєструємо помилку за допомогою інструменту на кшталт Honeybadger для моніторингу та налагодження. Якщо результат успішний (ProductRecommendations), ми транслюємо подію :new_recommendations за допомогою бібліотеки публікації/підписки Wisper, передаючи користувача та згенеровані рекомендації.

Використовуючи методи функціонального програмування, ми можемо створити модульний та підтримуваний обробник рекомендацій продуктів. Кожен крок є самодостатнім і може бути легко протестований, модифікований або замінений без впливу на загальний потік. Використання зіставлення зі зразком та класу Result допомагає нам елегантно обробляти помилки та гарантує, що обробник швидко завершує роботу, якщо будь-який крок стикається з проблемою.

Звичайно, це спрощений приклад, і в реальному сценарії вам потрібно буде інтегруватися з вашою платформою електронної комерції, обробляти крайові випадки і навіть зануритися в реалізацію алгоритмів рекомендацій. Однак основні принципи декомпозиції проблеми на менші кроки та використання методів функціонального програмування залишаються незмінними.

Виявлення шахрайства

Ось спрощений приклад того, як можна реалізувати обробник виявлення шахрайства, використовуючи той самий стиль програмування, орієнтованого на залізницю (ROP), в Ruby:

```ruby
class FraudDetectionWorker
  include Wisper::Publisher

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

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

Клас FraudDetection є *об'єктом-значенням*, який інкапсулює стан виявлення шахрайства для певної транзакції. Він забезпечує структурований спосіб аналізу та оцінки ризику шахрайства, пов'язаного з транзакцією, на основі різних факторів ризику.

```
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
```

Клас FraudDetection має такі атрибути:

- transaction: Посилання на транзакцію, яка аналізується на предмет шахрайства.

- risk_factors: Масив, який зберігає фактори ризику, пов'язані з транзакцією. Кожен фактор ризику представлений як хеш, де ключ - це опис фактора ризику, а значення - ймовірність шахрайства, пов'язана з цим фактором ризику.

Метод add_risk_factor дозволяє додавати фактор ризику до масиву risk_factors. Він приймає два параметри: description, який є рядком, що описує фактор ризику, та probability, який є числом з плаваючою комою, що представляє ймовірність шахрайства, пов'язану з цим фактором ризику. Ми використовуємо умовний оператор case..in для простої перевірки типів.

Метод high_risk?, який буде перевірятися в кінці ланцюжка, є предикатним методом, який порівнює fraud_probability (розраховану шляхом сумування ймовірностей усіх факторів ризику) з RISK_THRESHOLD.

Клас FraudDetection забезпечує чистий та інкапсульований спосіб управління виявленням шахрайства для транзакції. Він дозволяє додавати декілька факторів ризику, кожен зі своїм описом та ймовірністю, і надає метод для визначення, чи вважається транзакція високоризиковою на основі розрахованої ймовірності шахрайства. Клас може бути легко інтегрований у більшу систему виявлення шахрайства, де різні компоненти можуть співпрацювати для оцінки та зменшення ризику шахрайських транзакцій.

Нарешті, оскільки це все-таки книга про програмування з використанням ШІ, ось приклад реалізації класу CheckCustomerHistory з використанням ШІ-обробки за допомогою модуля ChatCompletion моєї бібліотеки Raix:

```
class CheckCustomerHistory
  include Raix::ChatCompletion

  attr_accessor :fraud_detection

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

    Consider factors such as the frequency of transactions, transaction
    amounts, geographical locations, and any deviations from the customer's
    typical behavior to generate a probability score as a float in the range
    of 0 to 1 (with 1 being absolute certainty of fraud).

    Output the results of your analysis, highlighting any red flags or areas
    of concern in the following JSON format:

    { description: <Summary of your findings>, probability: <Float> }
  END

  def self.check(fraud_detection)
    new(fraud_detection).call
  end

  def call
    chat_completion(json: true).tap do |result|
      fraud_detection.add_risk_factor(**result)
    end
    Result.ok(fraud_detection)
  rescue StandardError => e
    Result.err(FraudDetectionError.new(e))
  end

  private

  def initialize(fraud_detection)
    self.fraud_detection = fraud_detection
  end

  def transcript
```

```
43      tx_history = fraud_detection.transaction.user.tx_history
44      [
45        { system: INSTRUCTION },
46        { user: "Transaction history: #{tx_history.to_json}" },
47        { assistant: "OK. Please provide the current transaction." },
48        { user: "Current transaction: #{fraud_detection.transaction.to_json}" }
49      ]
50    end
51  end
```

У цьому прикладі, CheckCustomerHistory визначає константу INSTRUCTION, яка надає конкретні інструкції моделі ШІ щодо того, як аналізувати історію транзакцій клієнта на предмет потенційних індикаторів шахрайства через системну директиву

Метод self.check є методом класу, який ініціалізує новий екземпляр CheckCustomerHistory з об'єктом fraud_detection та викликає метод call для виконання аналізу історії клієнта.

У методі call отримується історія транзакцій клієнта та форматується у транскрипт, який передається моделі ШІ. Модель ШІ аналізує історію транзакцій на основі наданих інструкцій та повертає підсумок своїх висновків.

Висновки додаються до об'єкта fraud_detection, і оновлений об'єкт fraud_detection повертається як успішний Result.

Використовуючи модуль ChatCompletion, клас CheckCustomerHistory може застосовувати можливості ШІ для аналізу історії транзакцій клієнта та виявлення потенційних індикаторів шахрайства. Це дозволяє застосовувати більш складні та адаптивні методи виявлення шахрайства, оскільки модель ШІ може навчатися та адаптуватися до нових шаблонів та аномалій з часом.

Оновлений FraudDetectionWorker та клас CheckCustomerHistory демонструють, як обробники на основі ШІ можуть бути безперешкодно інтегровані, покращуючи процес виявлення шахрайства за допомогою інтелектуального аналізу та можливостей прийняття рішень.

Аналіз настроїв клієнтів

Ось ще один подібний приклад того, як можна реалізувати обробник аналізу настроїв клієнтів. Цього разу набагато менше пояснень, оскільки ви вже повинні розуміти, як працює цей стиль програмування:

```
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
```

У цьому прикладі CustomerSentimentAnalysisWorker кроки включають попередню обробку відгуків (наприклад, видалення шуму, токенізацію), виконання аналізу настроїв для визначення загального настрою (позитивного, негативного чи нейтрального), витягнення ключових фраз і тем, виявлення тенденцій і закономірностей та генерування практичних висновків на основі аналізу.

Застосування в охороні здоров’я

У галузі охорони здоров’я ШІ-працівники можуть допомагати медичним працівникам та дослідникам у різноманітних завданнях, що призводить до покращення результатів лікування пацієнтів та прискорення медичних відкриттів. Ось деякі приклади:

Прийом пацієнтів

ШІ-працівники можуть оптимізувати процес прийому пацієнтів шляхом автоматизації різних завдань та надання інтелектуальної допомоги.

Планування прийомів: ШІ-працівники можуть керувати плануванням прийомів, враховуючи уподобання пацієнтів, їхню доступність та терміновість їхніх медичних потреб. Вони можуть взаємодіяти з пацієнтами через розмовні інтерфейси, супроводжуючи їх через процес планування та знаходячи найбільш підходящий час прийому на основі вимог пацієнта та доступності медичного працівника.

Збір медичної історії: Під час прийому пацієнтів ШІ-працівники можуть допомагати у зборі та документуванні медичної історії пацієнта. Вони можуть вести інтерактивний діалог з пацієнтами, ставлячи відповідні запитання про їхні минулі захворювання, ліки, алергії та сімейний анамнез. ШІ-працівники можуть використовувати методи обробки природної мови для інтерпретації та структурування зібраної інформації, забезпечуючи її точне внесення до електронної медичної картки пацієнта.

Оцінка та стратифікація симптомів: ШІ-працівники можуть проводити початкову оцінку симптомів, запитуючи пацієнтів про їхні поточні симптоми, тривалість, тяжкість та пов’язані фактори. Використовуючи медичні бази знань та моделі машинного навчання, ці працівники можуть аналізувати надану

інформацію та генерувати попередні диференційні діагнози або рекомендувати відповідні наступні кроки, такі як планування консультації з медичним працівником або пропонування заходів самодопомоги.

Перевірка страхування: ШІ-працівники можуть допомагати з перевіркою страхування під час прийому пацієнтів. Вони можуть збирати дані про страховку пацієнта, спілкуватися зі страховими компаніями через API або веб-сервіси та перевіряти право на страхове покриття та пільги. Ця автоматизація допомагає оптимізувати процес перевірки страхування, зменшуючи адміністративне навантаження та забезпечуючи точність зібраної інформації.

Навчання пацієнтів та інструкції: ШІ-працівники можуть надавати пацієнтам відповідні навчальні матеріали та інструкції на основі їхніх конкретних медичних станів або майбутніх процедур. Вони можуть надавати персоналізований контент, відповідати на поширені запитання та надавати рекомендації щодо підготовки до прийому, інструкцій щодо прийому ліків або післялікувального догляду. Це допомагає тримати пацієнтів поінформованими та залученими протягом усього процесу лікування.

Використовуючи ШІ-працівників при прийомі пацієнтів, медичні організації можуть підвищити ефективність, зменшити час очікування та покращити загальний досвід пацієнтів. Ці працівники можуть виконувати рутинні завдання, збирати точну інформацію та надавати персоналізовану допомогу, дозволяючи медичним працівникам зосередитися на наданні якісної допомоги пацієнтам.

Оцінка ризиків пацієнтів

ШІ-працівники можуть відігравати вирішальну роль в оцінці ризиків пацієнтів шляхом аналізу різних джерел даних та застосування передових аналітичних методів.

Інтеграція даних: ШІ-працівники можуть збирати та аналізувати дані пацієнтів з різних джерел, таких як електронні медичні картки (ЕМК), медична візуалізація,

лабораторні результати, носимі пристрої та соціальні детермінанти здоров'я. Об'єднуючи цю інформацію в комплексний профіль пацієнта, ШІ-працівники можуть надати цілісне уявлення про стан здоров'я пацієнта та фактори ризику.

Стратифікація ризиків: ШІ-працівники можуть використовувати прогнозні моделі для стратифікації пацієнтів за різними категоріями ризику на основі їхніх індивідуальних характеристик та даних про здоров'я. Така стратифікація ризиків дозволяє медичним працівникам приділяти першочергову увагу пацієнтам, які потребують більш негайної уваги або втручання. Наприклад, пацієнти, визначені як такі, що мають високий ризик певного захворювання, можуть бути позначені для більш ретельного моніторингу, профілактичних заходів або раннього втручання.

Персоналізовані профілі ризику: ШІ-працівники можуть створювати персоналізовані профілі ризику для кожного пацієнта, виділяючи конкретні фактори, що впливають на їхні показники ризику. Ці профілі можуть включати інформацію про спосіб життя пацієнта, генетичну схильність, фактори навколишнього середовища та соціальні детермінанти здоров'я. Надаючи детальний аналіз факторів ризику, ШІ-працівники можуть допомогти медичним працівникам адаптувати стратегії профілактики та плани лікування до індивідуальних потреб пацієнтів.

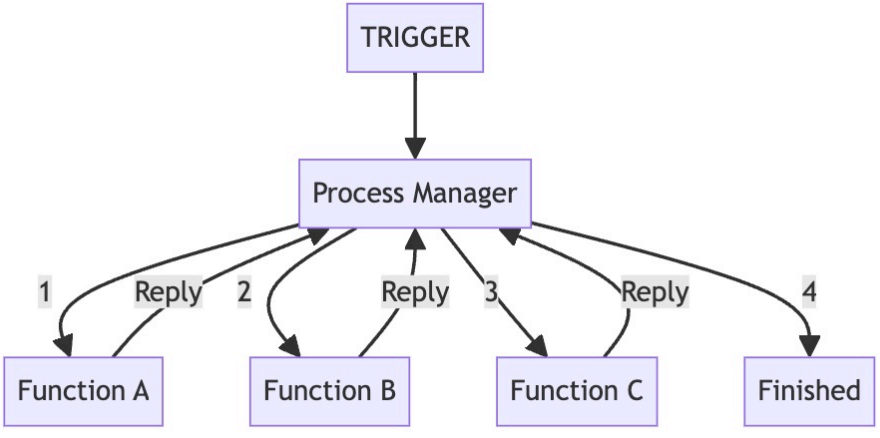
Постійний моніторинг ризиків: ШІ-працівники можуть постійно відстежувати дані пацієнтів та оновлювати оцінки ризиків у реальному часі. Коли з'являється нова інформація, така як зміни життєвих показників, результати аналізів або дотримання режиму прийому ліків, ШІ-працівники можуть перераховувати показники ризику та попереджати медичних працівників про будь-які значні зміни. Такий проактивний моніторинг дозволяє своєчасно втручатися та коригувати плани лікування пацієнтів.

Підтримка прийняття клінічних рішень: ШІ-працівники можуть інтегрувати результати оцінки ризиків у системи підтримки прийняття клінічних рішень,

надаючи медичним працівникам рекомендації та попередження на основі доказової медицини. Наприклад, якщо показник ризику пацієнта щодо певного стану перевищує певний поріг, ШІ-працівник може запропонувати медичному працівнику розглянути конкретні діагностичні тести, профілактичні заходи або варіанти лікування на основі клінічних рекомендацій та найкращих практик.

Ці робочі модулі можуть обробляти величезні обсяги даних пацієнтів, застосовувати складну аналітику та генерувати практичні висновки для підтримки прийняття клінічних рішень. Це в кінцевому підсумку призводить до покращення результатів лікування пацієнтів, зменшення витрат на охорону здоров'я та вдосконалення управління здоров'ям населення.

AI-працівник як менеджер процесів



У контексті застосунків на основі ШІ, робочий модуль може бути спроектований для функціонування як Менеджер процесів, як описано в книзі "Enterprise Integration Patterns" авторства Gregor Hohpe. Менеджер процесів

- це центральний компонент, який підтримує стан процесу та визначає наступні кроки обробки на основі проміжних результатів.

Коли AI-працівник діє як Менеджер процесів, він отримує вхідне повідомлення, яке ініціалізує процес, відоме як *тригерне повідомлення*. Потім AI-працівник підтримує стан виконання процесу (як протокол розмови) і обробляє повідомлення через серію кроків обробки, реалізованих як інструментальні функції, які можуть бути послідовними або паралельними, і викликаються на його розсуд.

Якщо ви використовуєте клас AI-моделей на кшталт GPT-4, який вміє виконувати функції паралельно, то ваш робочий модуль може виконувати кілька кроків одночасно. Щиро кажучи, я сам цього не пробував, і моя інтуїція підказує, що результати можуть бути різними.

Після кожного окремого кроку обробки керування повертається назад до AI-працівника, дозволяючи йому визначити наступний крок(и) обробки на основі поточного стану та отриманих результатів.

Зберігайте свої тригерні повідомлення

З мого досвіду, розумно реалізовувати тригерне повідомлення як об'єкт на основі бази даних. Таким чином кожен екземпляр процесу ідентифікується унікальним первинним ключем і надає місце для зберігання стану, пов'язаного з виконанням, включаючи протокол розмови AI.

Наприклад, ось спрощена версія класу моделі AccountChange Olympia, який представляє запит на внесення змін до облікового запису користувача.

```
# == 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
```

Клас AccountChange служить тригерним повідомленням, яке ініціює процес обробки запиту на зміну облікового запису. Зверніть увагу, як він транслюється

до підсистеми публікації/підписки Olympia на основі Wisper після завершення транзакції створення.

Збереження тригерного повідомлення в базі даних таким чином забезпечує постійний запис кожного запиту на зміну облікового запису. Кожному екземпляру класу AccountChange призначається унікальний первинний ключ, що дозволяє легко ідентифікувати та відстежувати окремі запити. Це особливо корисно для цілей аудит логування, оскільки дозволяє системі зберігати історичний запис усіх змін облікового запису, включаючи час подання запиту, які зміни були запитані та поточний стан кожного запиту.

У наведеному прикладі клас AccountChange містить такі поля, як description для фіксації деталей запитуваної зміни, state для представлення поточного стану запиту (наприклад, requested, complete, requires_human_review) та transcript для зберігання стенограми розмови зі ШІ щодо запиту. Поле description є фактичним запитом, який використовується для ініціювання першого чат-завершення зі ШІ. Збереження цих даних надає цінний контекст і дозволяє краще відстежувати та аналізувати процес зміни облікового запису.

Збереження тригерних повідомлень у базі даних забезпечує надійну обробку помилок та відновлення. Якщо під час обробки запиту на зміну облікового запису виникає помилка, система позначає запит як невдалий і переводить його в стан, що потребує втручання людини. Це гарантує, що жоден запит не буде втрачено чи забуто, і будь-які проблеми можуть бути належним чином розглянуті та вирішені.

ШІ-працівник, як Менеджер Процесів, забезпечує центральну точку контролю та надає потужні можливості звітування та налагодження процесів. Однак важливо зазначити, що використання ШІ-працівника як Менеджера Процесів для кожного сценарію робочого процесу у вашому додатку може бути надмірним.

Інтеграція ШІ-працівників у архітектуру вашого додатку

При включенні ШІ-працівників до архітектури вашого додатку необхідно врахувати кілька технічних міркувань для забезпечення плавної інтеграції та ефективної комунікації між ШІ-працівниками та іншими компонентами додатку. Цей розділ розглядає ключові аспекти проектування цих інтерфейсів, обробки потоку даних та управління життєвим циклом ШІ-працівників.

Проектування чітких інтерфейсів та протоколів зв'язку

Для забезпечення безперебійної інтеграції між ШІ-працівниками та іншими компонентами додатку важливо визначити чіткі інтерфейси та протоколи зв'язку. Розгляньте такі підходи:

API-інтеграція: Надання функціональності ШІ-працівників через чітко визначені API, такі як RESTful endpoints або GraphQL схеми. Це дозволяє іншим компонентам взаємодіяти з ШІ-працівниками за допомогою стандартних HTTP-запитів та відповідей. API-інтеграція забезпечує чіткий контракт між ШІ-працівниками та компонентами, що їх використовують, полегшуючи розробку, тестування та підтримку точок інтеграції.

Обмін повідомленнями: Реалізація шаблонів комунікації на основі повідомлень, таких як черги повідомлень або системи публікації-підписки, для забезпечення асинхронної взаємодії між ШІ-працівниками та іншими компонентами. Цей підхід відокремлює ШІ-працівників від решти додатку, забезпечуючи кращу масштабованість, відмовостійкість та слабке зв'язування. Обмін повідомленнями особливо корисний, коли обробка, що виконується ШІ-працівниками, є тривалою або ресурсомісткою, оскільки дозволяє іншим частинам додатку продовжувати виконання, не чекаючи завершення завдань ШІ-працівників.

Подієво-орієнтована архітектура: Проектування вашої системи навколо подій та тригерів, які активують ШІ-працівників при виконанні певних умов. ШІ-працівники можуть підписуватися на відповідні події та реагувати відповідним чином, виконуючи свої призначені завдання при виникненні подій. Подієво-орієнтована архітектура забезпечує обробку в реальному часі та дозволяє викликати ШІ-працівників за потреби, зменшуючи непотрібне споживання ресурсів. Цей підхід добре підходить для сценаріїв, де ШІ-працівники повинні реагувати на конкретні дії або зміни в стані додатку.

Обробка потоку даних та синхронізація

При інтеграції ШІ-працівників у ваш додаток важливо забезпечити плавний потік даних та підтримувати узгодженість даних між ШІ-працівниками та іншими компонентами. Розгляньте такі аспекти:

Підготовка даних: Перед подачею даних до ШІ-працівників може знадобитися виконання різних завдань з підготовки даних, таких як очищення, форматування та/або перетворення вхідних даних. Ви не тільки хочете переконатися, що ШІ-працівники можуть ефективно обробляти дані, але й хочете переконатися, що ви не витрачаєте токени, приділяючи увагу інформації, яку працівник може вважати в кращому випадку марною, а в гіршому - відволікаючою. Підготовка даних може включати такі завдання, як видалення шуму, обробка відсутніх значень або перетворення типів даних.

Збереження даних: Як ви будете зберігати та підтримувати дані, що проходять через ШІ-працівників? Враховуйте такі фактори, як обсяг даних, шаблони запитів та масштабованість. Чи потрібно зберігати стенограму ШІ як відображення його "процесу мислення" для цілей аудиту чи налагодження, чи достатньо мати запис лише результатів?

Отримання даних: Отримання даних, необхідних для працівників, може включати запити до баз даних, читання файлів або доступ до зовнішніх API.

Переконайтеся, що ви врахували латентність і те, як працівники зі штучним інтелектом матимуть доступ до найактуальніших даних. Чи потрібен їм повний доступ до вашої бази даних, чи варто визначити обсяг їхнього доступу вузько відповідно до їхніх завдань? А як щодо масштабування? Розгляньте механізми кешування для покращення продуктивності та зменшення навантаження на основні джерела даних.

Синхронізація даних: Коли кілька компонентів, включаючи працівників зі штучним інтелектом, отримують доступ і модифікують спільні дані, важливо впровадити належні механізми синхронізації для підтримки узгодженості даних. Стратегії блокування бази даних, такі як оптимістичне чи песимістичне блокування, можуть допомогти запобігти конфліктам і забезпечити цілісність даних. Впровадьте методи управління транзакціями для групування пов'язаних операцій з даними та підтримки властивостей ACID (атомарність, узгодженість, ізольованість та довговічність)

Обробка помилок та відновлення: Впровадьте надійні механізми обробки помилок та відновлення для вирішення проблем, пов'язаних з даними, які можуть виникнути під час процесу потоку даних. Обробляйте винятки коректно та надавайте змістовні повідомлення про помилки для полегшення налагодження. Впровадьте механізми повторних спроб та стратегії відмовостійкості для обробки тимчасових збоїв або мережевих порушень. Визначте чіткі процедури відновлення даних у випадку пошкодження або втрати даних.

Ретельно проектуючи та впроваджуючи механізми потоку та синхронізації даних, ви можете забезпечити доступ працівників зі штучним інтелектом до точних, узгоджених та актуальних даних. Це дозволяє їм ефективно виконувати свої завдання та отримувати надійні результати.

Керування життєвим циклом працівників зі штучним інтелектом

Розробіть стандартизований процес ініціалізації та налаштування працівників зі штучним інтелектом. Я віддаю перевагу ƒреймворкам, які стандартизують спосіб визначення налаштувань, таких як назви моделей, системні директиви та визначення ƒункцій. Переконайтеся, що процес ініціалізації автоматизований та відтворюваний для полегшення розгортання та масштабування.

Впровадьте комплексні механізми моніторингу та логування для відстеження працездатності та продуктивності працівників зі штучним інтелектом. Збирайте метрики, такі як використання ресурсів, час обробки, частота помилок та пропускна здатність. Використовуйте централізовані системи логування, такі як ELK stack (Elasticsearch, Logstash, Kibana) для агрегації та аналізу логів від кількох працівників зі штучним інтелектом.

Вбудуйте відмовостійкість та стійкість в архітектуру працівників зі штучним інтелектом. Впровадьте механізми обробки помилок та відновлення для коректної обробки збоїв або винятків. Великі мовні моделі все ще є передовою технологією; постачальники часто несподівано виходять з ладу. Використовуйте механізми повторних спроб та переривники для запобігання каскадним збоям.

Компонованість та оркестрація працівників зі штучним інтелектом

Однією з ключових переваг архітектури працівників зі штучним інтелектом є її компонованість, яка дозволяє комбінувати та оркеструвати кількох працівників зі штучним інтелектом для вирішення складних завдань. Розбиваючи більше завдання на менші, більш керовані підзавдання, кожне з яких обробляється спеціалізованим працівником зі штучним інтелектом, ви можете створювати

потужні та гнучкі системи. У цьому розділі ми розглянемо різні підходи до компонування та оркестрації "множини" працівників зі штучним інтелектом.

Ланцюжок працівників зі штучним інтелектом для багатоетапних робочих процесів

У багатьох сценаріях складне завдання можна розкласти на серію послідовних кроків, де вихідні дані одного працівника зі штучним інтелектом стають вхідними даними для наступного. Це з'єднання працівників зі штучним інтелектом створює багатоетапний робочий процес або конвеєр. Кожен працівник зі штучним інтелектом у ланцюжку зосереджується на конкретному підзавданні, а кінцевий результат є результатом спільних зусиль усіх працівників.

Розглянемо приклад у контексті додатка Ruby on Rails для обробки користувацького контенту. Робочий процес включає наступні кроки, які, варто визнати, ймовірно, кожен занадто простий, щоб варто було розкладати таким чином у реальних випадках використання, але вони роблять приклад легшим для розуміння:

1. **Очищення тексту**: Працівник зі штучним інтелектом, відповідальний за видалення HTML-тегів, перетворення тексту в нижній регістр та обробку нормалізації Unicode.

2. **Визначення мови**: Працівник зі штучним інтелектом, який визначає мову очищеного тексту.

3. **Аналіз тональності**: Працівник зі штучним інтелектом, який визначає тональність (позитивну, негативну або нейтральну) тексту на основі визначеної мови.

4. **Категоризація контенту**: Працівник зі штучним інтелектом, який класифікує текст за попередньо визначеними категоріями, використовуючи методи обробки природної мови.

Ось дуже спрощений приклад того, як можна з'єднати цих працівників зі штучним інтелектом разом, використовуючи 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
```

У цьому прикладі клас ContentProcessor ініціалізується з необробленим текстом і об'єднує ШІ-працівників у ланцюжок у методі process. Кожен ШІ-працівник виконує своє конкретне завдання та передає результат наступному працівнику в ланцюжку. Кінцевий результат - це хеш, що містить очищений текст, виявлену мову, тональність та категорію контенту.

Паралельна обробка для незалежних ШІ-працівників

У попередньому прикладі ШІ-працівники об'єднані послідовно, де кожен працівник обробляє текст і передає результат наступному працівнику. Однак, якщо у вас є кілька ШІ-працівників, які можуть працювати незалежно з одним і тим же вхідним текстом, ви можете оптимізувати робочий процес, обробляючи їх паралельно.

У даному сценарії, після того як TextCleanupWorker виконав очищення тексту, LanguageDetectionWorker, SentimentAnalysisWorker та CategorizationWorker можуть всі одночасно обробляти очищений текст незалежно один від одного.

Запускаючи цих працівників паралельно, ви можете потенційно зменшити загальний час обробки та підвищити ефективність вашого робочого процесу.

Щоб досягти паралельної обробки в Ruby, ви можете використовувати методи паралельного виконання, такі як потоки або асинхронне програмування. Ось приклад того, як можна модифікувати клас ContentProcessor для паралельної обробки останніх трьох працівників за допомогою потоків:

```
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
```

У цій оптимізованій версії ми використовуємо бібліотеку concurrent-ruby для створення об'єктів Concurrent::Future для кожного з незалежних ШІ-обробників.

Об’єкт Future представляє обчислення, яке буде виконано асинхронно в окремому потоці.

Після етапу очищення тексту ми створюємо три об’єкти Future: language_future, sentiment_future та category_future. Кожен Future виконує відповідний ШІ-обробник (LanguageDetectionWorker, SentimentAnalysisWorker та CategorizationWorker) в окремому потоці, передаючи cleaned_text як вхідні дані.

Викликаючи метод value для кожного Future, ми очікуємо завершення обчислень і отримуємо результат. Метод value блокує виконання доти, доки результат не стане доступним, гарантуючи, що всі паралельні обробники завершили обробку перед продовженням.

Нарешті, ми створюємо вихідний хеш з очищеним текстом та результатами паралельних обробників, як і в оригінальному прикладі.

Обробляючи незалежні ШІ-обробники паралельно, можна потенційно зменшити загальний час обробки порівняно з послідовним виконанням. Ця оптимізація особливо корисна при роботі з ресурсомісткими завданнями або при обробці великих обсягів даних.

Проте важливо зазначити, що фактичний приріст продуктивності залежить від різних факторів, таких як складність кожного обробника, доступні системні ресурси та накладні витрати на керування потоками. Завжди рекомендується проводити тестування продуктивності та профілювання коду, щоб визначити оптимальний рівень паралелізму для вашого конкретного випадку.

Крім того, при реалізації паралельної обробки слід враховувати спільні ресурси або залежності між обробниками. Переконайтеся, що обробники можуть працювати незалежно без конфліктів або змагань за ресурси. Якщо є залежності або спільні ресурси, можливо, вам потрібно буде реалізувати відповідні механізми синхронізації для збереження цілісності даних та уникнення таких проблем, як взаємні блокування або неузгоджені результати.

Глобальне блокування інтерпретатора Ruby та асинхронна обробка

Важливо розуміти наслідки використання глобального блокування інтерпретатора Ruby (GIL) при розгляді асинхронної потокової обробки в Ruby.

GIL — це механізм в інтерпретаторі Ruby, який гарантує, що лише один потік може виконувати код Ruby одночасно, навіть на багатоядерних процесорах. Це означає, що хоча в процесі Ruby можна створювати та керувати кількома потоками, лише один потік може активно виконувати код Ruby в будь-який момент часу.

GIL розроблений для спрощення реалізації інтерпретатора Ruby та забезпечення потокової безпеки для внутрішніх структур даних Ruby. Однак він також обмежує можливості справжнього паралельного виконання коду Ruby.

Коли ви використовуєте потоки в Ruby, наприклад, за допомогою бібліотеки concurrent-ruby або вбудованого класу Thread, потоки підпорядковуються обмеженням GIL. GIL дозволяє кожному потоку виконувати код Ruby протягом короткого проміжку часу перед переключенням на інший потік, створюючи ілюзію паралельного виконання.

Однак через GIL фактичне виконання коду Ruby залишається послідовним. Поки один потік виконує код Ruby, інші потоки фактично призупинені, очікуючи своєї черги на отримання GIL та виконання.

Це означає, що потокова асинхронна обробка в Ruby найефективніша для задач, пов'язаних з введенням/виведенням, таких як очікування відповідей від зовнішніх API (наприклад, великих мовних моделей, розміщених на сторонніх серверах) або виконання операцій введення/виведення файлів. Коли

потік стикається з операцією введення/виведення, він може звільнити GIL, дозволяючи іншим потокам виконуватися під час очікування завершення введення/виведення.

З іншого боку, для задач з інтенсивним використанням процесора, таких як інтенсивні обчислення або тривала обробка ШІ-обробниками, GIL може обмежувати потенційний приріст продуктивності потокового паралелізму. Оскільки лише один потік може виконувати код Ruby одночасно, загальний час виконання може не значно зменшитися порівняно з послідовною обробкою.

Для досягнення справжнього паралельного виконання задач з інтенсивним використанням процесора в Ruby, можливо, вам доведеться розглянути альтернативні підходи, такі як:

- Використання паралелізму на рівні процесів з кількома процесами Ruby, кожен з яких працює на окремому ядрі процесора.
- Використання зовнішніх бібліотек або фреймворків, які надають нативні розширення або інтерфейси до мов без GIL, таких як C або Rust.,
- Використання розподілених обчислювальних фреймворків або черг повідомлень для розподілу задач між кількома машинами або процесами.

Важливо враховувати характер ваших задач та обмеження, що накладаються GIL, при проектуванні та реалізації асинхронної обробки в Ruby. Хоча потокова асинхронна обробка може надавати переваги для задач з інтенсивним введенням/виведенням, вона може не забезпечувати значного покращення продуктивності для задач з інтенсивним використанням процесора через обмеження GIL.

Ансамблеві методи для підвищення точності

Ансамблеві методи передбачають об'єднання результатів роботи кількох ШІ-обробників для підвищення загальної точності або надійності системи. Замість того, щоб покладатися на один ШІ-обробник, ансамблеві методи використовують колективний інтелект кількох обробників для прийняття більш обґрунтованих рішень.

Ансамблі особливо важливі, якщо різні частини вашого робочого процесу найкраще працюють з різними моделями ШІ, що трапляється частіше, ніж можна подумати. Потужні моделі на кшталт GPT-4 є надзвичайно дорогими порівняно з менш потужними варіантами з відкритим кодом, і ймовірно, вони не потрібні для кожного окремого кроку робочого процесу вашого застосунку.

Одним із поширених методів ансамблювання є мажоритарне голосування, коли кілька ШІ-працівників незалежно обробляють один і той самий вхідний матеріал, а кінцевий результат визначається більшістю голосів. Цей підхід може допомогти зменшити вплив помилок окремих працівників і покращити загальну надійність системи.

Розглянемо приклад, де у нас є три ШІ-працівники для аналізу тональності, кожен з яких використовує різну модель або має різний контекст. Ми можемо об'єднати їхні результати за допомогою мажоритарного голосування, щоб визначити остаточний прогноз тональності.

```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
```

У цьому прикладі клас SentimentAnalysisEnsemble ініціалізується з текстом і викликає трьох різних ШІ-працівників для аналізу настроїв. Метод analyze збирає прогнози від кожного працівника та визначає переважаючий настрій за допомогою методів group_by та max_by. Кінцевим результатом є настрій, який отримує найбільше голосів від ансамблю працівників

Ансамблі - це явно той випадок, коли експериментування з паралелізмом може бути варте ваших зусиль.

Динамічний вибір та виклик ШІ-працівників

У деяких, якщо не в більшості випадків, конкретний ШІ-працівник, якого потрібно викликати, може залежати від умов виконання або користувацьких введень. Динамічний вибір та виклик ШІ-працівників забезпечують гнучкість та адаптивність системи.

Ви можете відчути спокусу спробувати вмістити багато функціональності в одного ШІ-працівника, надавши йому багато функцій та великий складний промпт, який пояснює, як їх викликати. Не піддавайтеся цій спокусі, повірте мені. Одна з причин, чому підхід, який ми обговорюємо в цьому розділі, називається “Множина Працівників”, полягає в тому, щоб нагадати нам, що бажано мати багато спеціалізованих працівників, кожен з яких виконує свою маленьку роботу для досягнення більшої мети.

Наприклад, розглянемо застосунок чат-бота, де різні ШІ-працівники відповідають за обробку різних типів користувацьких запитів. На основі введення користувача, застосунок динамічно обирає відповідного ШІ-працівника для обробки запиту.

```
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
```

У цьому прикладі ChatbotController отримує запит користувача через дію process_query. Спочатку він використовує QueryClassifierWorker для визначення

типу запиту. На основі класифікованого типу запиту контролер динамічно обирає відповідний ШІ-обробник для генерації відповіді. Такий динамічний вибір дозволяє чатботу обробляти різні типи запитів та спрямовувати їх до відповідних ШІ-обробників.

Оскільки робота QueryClassifierWorker є відносно простою і не потребує багато контексту чи визначень функцій, ви, ймовірно, можете реалізувати її за допомогою надшвидкої малої ВММ, як-от mistralai/mixtral-8x7b-instruct:nitro. Її можливості наближаються до рівня GPT-4 у багатьох завданнях, і на момент написання цього тексту Groq може обслуговувати її з вражаючою швидкістю 444 токени на секунду.

Поєднання традиційної ОПМ з ВММ

Хоча Великі мовні моделі (ВММ) здійснили революцію в галузі обробки природної мови (ОПМ), пропонуючи неперевершену універсальність та продуктивність у широкому спектрі завдань, вони не завжди є найефективнішим або найбільш економічним рішенням для кожної проблеми. У багатьох випадках поєднання традиційних методів ОПМ з ВММ може призвести до більш оптимізованих, цільових та економічних підходів до вирішення конкретних завдань ОПМ.

Думайте про ВММ як про швейцарські армійські ножі ОПМ — неймовірно універсальні та потужні, але не обов'язково найкращий інструмент для кожної роботи. Іноді спеціалізований інструмент, як-от штопор чи консервний ніж, може бути ефективнішим для конкретного завдання. Аналогічно, традиційні методи ОПМ, такі як кластеризація документів, ідентифікація тем та класифікація, часто можуть забезпечити більш цільові та економічно ефективні рішення для певних аспектів вашого конвеєра ОПМ.

Однією з ключових переваг традиційних методів ОПМ є їхня обчислювальна ефективність. Ці методи, які часто спираються на простіші статистичні моделі або підходи на основі правил, можуть обробляти великі обсяги текстових даних набагато швидше і з меншими обчислювальними витратами порівняно з ВММ. Це робить їх особливо придатними для завдань, що включають аналіз та організацію великих корпусів документів, таких як кластеризація подібних статей або визначення ключових тем у колекції текстів.

Крім того, традиційні методи ОПМ часто можуть досягти високої точності та влучності для конкретних завдань, особливо коли вони навчені на доменно-специфічних наборах даних. Наприклад, добре налаштований класифікатор документів, що використовує традиційні алгоритми машинного навчання, такі як Машини опорних векторів (МОВ) або Наївний Баєс, може точно категоризувати документи за попередньо визначеними категоріями з мінімальними обчислювальними витратами.

Проте ВММ справді відзначаються, коли йдеться про завдання, що вимагають глибшого розуміння мови, контексту та міркування. Їхня здатність генерувати зв'язний та контекстуально релевантний текст, відповідати на запитання та узагальнювати довгі уривки є неперевершеною порівняно з традиційними методами ОПМ. ВММ можуть ефективно обробляти складні лінгвістичні явища, такі як неоднозначність, кореференція та ідіоматичні вирази, що робить їх незамінними для завдань, які вимагають генерації або розуміння природної мови.

Справжня сила полягає в поєднанні традиційних методів ОПМ з ВММ для створення гібридних підходів, які використовують переваги обох. Використовуючи традиційні методи ОПМ для завдань, таких як попередня обробка документів, кластеризація та витяг тем, ви можете ефективно організувати та структурувати ваші текстові дані. Ця структурована інформація потім може подаватися у ВММ для більш складних завдань, таких як генерація

узагальнень, відповіді на запитання або створення комплексних звітів.

Наприклад, розглянемо випадок використання, коли ви хочете згенерувати звіт про тренди для конкретної області на основі великого корпусу окремих документів про тренди. Замість того, щоб покладатися виключно на ВММ, що може бути обчислювально затратним і трудомістким для обробки великих обсягів тексту, ви можете застосувати гібридний підхід:

1. Використовуйте традиційні методи ОПМ, такі як тематичне моделювання (наприклад, Латентне розміщення Діріхле) або алгоритми кластеризації (наприклад, К-середніх), для групування подібних документів про тренди та визначення ключових тем у корпусі.
2. Подавайте кластеризовані документи та визначені теми у ВММ, використовуючи її перевершене розуміння мови та можливості генерації для створення зв'язних та інформативних узагальнень для кожного кластера або теми.
3. Нарешті, використовуйте ВММ для генерації комплексного звіту про тренди, поєднуючи окремі узагальнення, виділяючи найважливіші тренди та надаючи висновки й рекомендації на основі агрегованої інформації.

Поєднуючи традиційні методи ОПМ з ВММ таким чином, ви можете ефективно обробляти великі обсяги текстових даних, отримувати значущі висновки та генерувати високоякісні звіти, оптимізуючи при цьому обчислювальні ресурси та витрати.

Розпочинаючи проєкти з НЛП, важливо ретельно оцінити конкретні вимоги та обмеження кожного завдання та розглянути, як традиційні методи НЛП та ВММ можна поєднати для досягнення найкращих результатів. Поєднуючи ефективність і точність традиційних методів із універсальністю та потужністю ВММ, ви зможете створювати високоефективні та економічні рішення НЛП, які принесуть користь вашим користувачам та зацікавленим сторонам.

Використання інструментів

У сфері розробки застосунків на основі ШІ концепція “використання інструментів” або “виклик функцій” стала потужною технікою, яка дозволяє вашій ВММ підключатися до зовнішніх інструментів, API, функцій, баз даних та інших ресурсів. Цей підхід дозволяє реалізувати ширший набір поведінок, ніж просто виведення тексту, та забезпечує більш динамічну взаємодію між компонентами ШІ та рештою екосистеми вашого застосунку. Як ми розглянемо в цьому розділі, використання інструментів також надає вам можливість змусити вашу модель ШІ генерувати дані структурованим способом.

Що таке використання інструментів?

Використання інструментів, також відоме як виклик функцій, — це техніка, яка дозволяє розробникам визначати список функцій, з якими ВММ може

взаємодіяти під час процесу генерації. Ці інструменти можуть варіюватися від простих службових функцій до складних API чи запитів до баз даних. Надаючи ВММ доступ до цих інструментів, розробники можуть розширити можливості моделі та дозволити їй виконувати завдання, які потребують зовнішніх знань або дій.

Рисунок 8. Приклад визначення функції для ШІ-працівника, який аналізує документи

```
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
```

Ключова ідея використання інструментів полягає в тому, щоб надати LLM можливість динамічно вибирати та виконувати відповідні інструменти на основі введених даних користувача або поставленого завдання. Замість того,

щоб покладатися виключно на попередньо навчені знання моделі, використання інструментів дозволяє LLM застосовувати зовнішні ресурси для генерації більш точних, релевантних та придатних для виконання відповідей. Використання інструментів робить такі методи, як RAG (Генерація з розширеним пошуком), значно простішими для впровадження, ніж це було б без них.

> Зауважте, що якщо не вказано інше, ця книга передбачає, що ваша модель ШІ не має доступу до жодних вбудованих інструментів на стороні сервера. Будь-які інструменти, які ви хочете зробити доступними для вашого ШІ, повинні бути явно оголошені вами в кожному API-запиті, з положеннями про їх виконання, якщо і коли ваш ШІ повідомить вам, що хотів би використати цей інструмент у своїй відповіді.

Потенціал використання інструментів

Використання інструментів відкриває широкий спектр можливостей для застосунків на базі ШІ. Ось кілька прикладів того, чого можна досягти за допомогою використання інструментів:

1. **Чат-боти та віртуальні помічники**: Підключаючи LLM до зовнішніх інструментів, чат-боти та віртуальні помічники можуть виконувати складніші завдання, такі як отримання інформації з баз даних, виконання API-викликів або взаємодія з іншими системами. Наприклад, чат-бот може використовувати інструмент CRM для зміни статусу угоди на основі запиту користувача.
2. **Аналіз даних та інсайти**: LLM можуть бути підключені до інструментів або бібліотек аналізу даних для виконання складних завдань з обробки

даних. Це дозволяє застосункам генерувати інсайти, проводити порівняльний аналіз або надавати рекомендації на основі даних відповідно до запитів користувачів.

3. **Пошук та отримання інформації**: Використання інструментів дозволяє LLM взаємодіяти з пошуковими системами, векторними базами даних або іншими системами отримання інформації. Перетворюючи запити користувачів на пошукові запити, LLM може отримувати релевантну інформацію з різних джерел і надавати вичерпні відповіді на запитання користувачів.
4. **Інтеграція із зовнішніми сервісами**: Використання інструментів забезпечує безперебійну інтеграцію між застосунками на базі ШІ та зовнішніми сервісами або API. Наприклад, LLM може взаємодіяти з API погоди для надання оновлень про погоду в реальному часі або з API перекладу для генерації багатомовних відповідей.

Робочий процес використання інструментів

Робочий процес використання інструментів зазвичай включає чотири ключові кроки:

1. Включення визначень функцій у контекст запиту
2. Динамічний (або явний) вибір інструменту
3. Виконання функції(й)
4. Необов’язкове продовження початкового запиту

Розглянемо кожен із цих кроків детально.

Включення визначень функцій у контекст запиту

ШІ знає, які інструменти є в його розпорядженні, тому що ви надаєте йому список як частину вашого запиту на завершення (зазвичай визначається як функції з використанням варіанту схеми JSON).

Точний синтаксис визначення інструменту залежить від конкретної моделі.

Ось як визначається функція get_weather в Claude 3:

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

І ось як ви визначили б ту саму функцію для GPT-4, передаючи її як значення параметра tools:

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

> Майже те саме, тільки чомусь по-іншому! Як дратівливо.

Визначення функцій задають назву, опис та вхідні параметри. Вхідні параметри можна додатково визначати за допомогою атрибутів, таких як переліки для обмеження допустимих значень, а також вказувати, чи є параметр обов'язковим чи ні.

Окрім власне визначень функцій, ви також можете включати інструкції або контекст щодо того, чому і як використовувати функцію в системній директиві.

Наприклад, мій інструмент веб-пошуку в Olympia містить таку системну директиву, яка нагадує ШІ, що він має у своєму розпорядженні згадані інструменти:

```
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.
```

Надання детальних описів вважається найважливішим фактором продуктивності інструменту. Ваші описи повинні пояснювати кожну деталь про інструмент, включаючи:

- Що робить інструмент
- Коли його слід використовувати (а коли ні)
- Що означає кожен параметр і як він впливає на поведінку інструменту
- Будь-які важливі застереження або обмеження, що стосуються реалізації інструменту

Чим більше контексту ви можете надати ШІ про ваші інструменти, тим краще він зможе вирішувати, коли і як їх використовувати. Наприклад, Anthropic рекомендує щонайменше 3-4 речення для опису кожного інструменту для своєї серії Claude 3, або більше, якщо інструмент складний.

Це може бути неінтуїтивно, але описи також вважаються важливішими за приклади. Хоча ви можете включити приклади використання інструменту в його опис або в супровідний промпт, це менш важливо, ніж мати чітке та вичерпне пояснення призначення та параметрів інструменту. Додавайте приклади лише після того, як ви повністю розробили опис.

Ось приклад специфікації API-функції в стилі Stripe:

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

```
            }
          },
          "required": [ "type", "number" ]
        }
      },
      "required": [ "name", "email", "identification" ]
    }
  }
}
```

На практиці деякі моделі мають проблеми з обробкою вкладених специфікацій функцій та складних типів вихідних даних, таких як масиви, словники тощо. Але теоретично ви повинні мати можливість надавати специфікації схеми JSON довільної глибини!

Динамічний вибір інструментів

Коли ви виконуєте чат-доповнення, що включає визначення інструментів, BMM динамічно обирає найбільш відповідний інструмент(и) для використання та генерує необхідні вхідні параметри для кожного інструменту.

На практиці здатність ШІ викликати *саме* потрібну функцію та *точно* дотримуватися вашої специфікації для вхідних даних є непередбачуваною. Зниження гіперпараметра температури до 0.0 значно допомагає, але з мого досвіду ви все одно будете отримувати випадкові помилки. Ці помилки включають галюциновані назви функцій, неправильно названі або просто відсутні вхідні параметри. Параметри передаються як JSON, що означає, що іноді ви побачите помилки, спричинені обрізаним, неправильно процитованим або іншим чином пошкодженим JSON.

Патерни Самовідновлюваних даних можуть допомогти автоматично виправити виклики функцій, що ламаються через синтаксичні помилки.

Примусовий (або Явний) вибір інструментів

Деякі моделі дають вам можливість примусово викликати певну функцію як параметр у запиті. В іншому випадку, рішення про виклик функції повністю залежить від розсуду ШІ.

Можливість примусового виклику функції є критично важливою в певних сценаріях, де ви хочете забезпечити виконання конкретного інструменту чи функції, незалежно від процесу динамічного вибору ШІ. Є кілька причин, чому ця можливість важлива:

1. **Явний контроль**: Ви можете використовувати ШІ як *Дискретний компонент* або в попередньо визначеному робочому процесі, який вимагає виконання конкретної функції в конкретний час. Примусовим викликом ви можете гарантувати, що бажана функція буде викликана замість того, щоб ввічливо просити ШІ це зробити.
2. **Налагодження та тестування**: При розробці та тестуванні застосунків на базі ШІ, можливість примусового виклику функцій є неоціненною для цілей налагодження. Явно викликаючи конкретні функції, ви можете ізолювати та протестувати окремі компоненти вашого застосунку. Це дозволяє перевірити правильність реалізації функцій, перевірити вхідні параметри та переконатися, що повертаються очікувані результати.
3. **Обробка крайових випадків**: Можуть бути крайові випадки або виняткові сценарії, де процес динамічного вибору ШІ може не вибрати виконання функції, яку слід виконати, і ви знаєте це на основі зовнішніх процесів. У таких випадках можливість примусового виклику функції дозволяє явно обробляти ці ситуації. Визначте правила або умови в логіці вашого застосунку, щоб визначити, коли перевизначати розсуд ШІ.
4. **Узгодженість та відтворюваність**: Якщо у вас є конкретна послідовність функцій, які потрібно виконати в певному порядку, примусові виклики

гарантують, що та сама послідовність виконується щоразу. Це особливо важливо в застосунках, де критичними є узгодженість та передбачувана поведінка, наприклад, у фінансових системах або наукових симуляціях.

5. **Оптимізація продуктивності**: У деяких випадках примусовий виклик функції може призвести до оптимізації продуктивності. Якщо ви знаєте, що конкретна функція потрібна для певного завдання, і що процес динамічного вибору ШІ може внести непотрібні накладні витрати, ви можете обійти процес вибору і безпосередньо викликати потрібну функцію. Це може допомогти зменшити затримку та покращити загальну ефективність вашого застосунку.

Підсумовуючи, можливість примусового виклику функцій у застосунках на базі ШІ забезпечує явний контроль, допомагає в налагодженні та тестуванні, обробляє крайові випадки, забезпечує узгодженість та відтворюваність. Це потужний інструмент у вашому арсеналі, але нам потрібно обговорити ще один аспект цієї важливої функції.

У багатьох випадках прийняття рішень ми завжди хочемо, щоб модель робила виклик функції і можемо ніколи не хотіти, щоб модель відповідала лише своїми внутрішніми знаннями. Наприклад, якщо ви маршрутизуєте між кількома моделями, спеціалізованими на різних завданнях (багатомовне введення, математика тощо), ви можете використовувати модель з викликом функцій для делегування запитів одній з допоміжних моделей і ніколи не відповідати самостійно.

Параметр вибору інструменту

GPT-4 та інші мовні моделі, які підтримують виклик функцій, дають вам параметр tool_choice для контролю над тим, чи потрібне використання інструменту як частина доповнення. Цей параметр має три можливі значення:

- auto надає ШІ повну свободу у використанні інструменту або простому реагуванні
- required вказує ШІ, що вона *повинна* викликати інструмент *замість* відповіді, але залишає вибір інструменту на розсуд ШІ
- Третій варіант - встановити параметр name_of_function, який ви хочете примусово викликати. Більше про це в наступному розділі.

Зверніть увагу, що якщо ви встановите tool choice як required, модель буде змушена вибрати найбільш відповідну функцію для виклику з-поміж наданих, навіть якщо жодна з них насправді не підходить для запиту. На момент публікації мені невідомо про модель, яка б повертала порожню відповідь tool_calls або якимось іншим чином повідомляла про те, що не знайшла відповідної функції для виклику.

Примусовий Виклик Функції для Отримання Структурованого Виводу

Можливість примусового виклику функції дає вам спосіб отримати структуровані дані з чат-завершення замість того, щоб самостійно витягувати їх з текстової відповіді.

Чому примусовий виклик функцій для отримання структурованого виводу - це така важлива справа? Просто кажучи, тому що витягування структурованих даних з виводу ВММ - це справжній головний біль. Ви можете трохи полегшити собі життя, запитуючи дані в форматі XML, але тоді вам доведеться парсити XML. І що робити, коли цей XML відсутній, тому що ваш

> ШІ відповів: “Вибачте, але я не можу згенерувати запитані дані, тому що бла, бла, бла...”

При використанні інструментів таким чином:

- Вам, ймовірно, слід визначити єдиний інструмент у вашому запиті
- Пам’ятайте про необхідність примусового використання його функції за допомогою параметра tool_choice.
- Пам’ятайте, що модель передасть вхідні дані інструменту, тому назва інструменту та опис повинні бути з перспективи моделі, а не вашої.

Останній пункт заслуговує на приклад для ясності. Припустімо, ви просите ШІ зробити аналіз тональності тексту користувача. Назва функції не була б analyze_sentiment, а скоріше щось на кшталт save_sentiment_analysis. ШІ - це той, хто робить аналіз тональності, *а не інструмент*. Все, що робить інструмент (з точки зору ШІ) - це зберігає результати аналізу.

Ось приклад використання Claude 3 для запису підсумку зображення у добре структурований JSON, цього разу з командного рядка за допомогою curl:

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

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

```
            "required": [ "key_colors", "description" ]
        }
    }],
    "messages": [
        {
            "role": "user",
            "content": [
                {
                    "type": "image",
                    "source": {
                        "type": "base64",
                        "media_type": "'$IMAGE_MEDIA_TYPE'",
                        "data": "'$IMAGE_BASE64'"
                    }
                },
                {
                    "type": "text",
                    "text": "Use `record_summary` to describe this image."
                }
            ]
        }
    ]
}'
```

У наведеному прикладі ми використовуємо модель Claude 3 від Anthropic для генерації структурованого JSON-підсумку зображення. Ось як це працює:

1. Ми визначаємо єдиний інструмент з назвою record_summary у масиві tools корисного навантаження запиту. Цей інструмент відповідає за запис підсумку зображення у добре структурований JSON.
2. Інструмент record_summary має input_schema, що визначає очікувану структуру JSON-виводу. Він визначає три властивості:

    - key_colors: Масив об'єктів, що представляють основні кольори зображення. Кожен об'єкт кольору має властивості для значень червоного, зеленого та синього (в діапазоні від 0.0 до 1.0) та зрозумілу людині назву кольору у форматі snake_case.

- description: Рядкова властивість для короткого опису зображення, обмеженого 1-2 реченнями.
- estimated_year: Необов'язкова цілочисельна властивість для приблизного року створення зображення, якщо воно виглядає як нехудожня фотографія.

3. У масиві messages ми надаємо дані зображення як рядок у кодуванні base64 разом із типом медіа. Це дозволяє моделі обробляти зображення як частину вхідних даних.
4. Ми також спонукаємо Claude використовувати інструмент record_summary для опису зображення.
5. Коли запит надсилається до моделі Claude 3, вона аналізує зображення та генерує JSON-підсумок на основі вказаної input_schema. Модель витягує основні кольори, надає короткий опис та оцінює рік створення зображення (якщо це можливо).
6. Згенерований JSON-підсумок передається як параметри до інструменту record_summary, забезпечуючи структуроване представлення ключових характеристик зображення.

Використовуючи інструмент record_summary з чітко визначеною input_schema, ми можемо отримати структурований JSON-підсумок зображення без покладання на видобування простого тексту. Цей підхід гарантує, що вивід дотримується послідовного формату і може бути легко проаналізований та оброблений компонентами низхідного потоку програми.

Можливість примусового виклику функції та визначення очікуваної структури виводу є потужною особливістю використання інструментів у програмах на основі ШІ. Це дозволяє розробникам мати більший контроль над згенерованим виводом та спрощує інтеграцію даних, згенерованих ШІ, у робочий процес їхньої програми.

Виконання функції(й)

Ви визначили функції та надали запит вашому ШІ, який вирішив, що має викликати одну з ваших функцій. Тепер час для вашого коду програми чи бібліотеки, якщо ви використовуєте Ruby gem на кшталт raix-rails, відправити виклик функції та її параметри до відповідної реалізації *у вашому коді програми.*

Ваш код програми вирішує, що робити з результатами виконання функції. Можливо, це включає один рядок коду в лямбді, або можливо це включає виклик зовнішнього API. Можливо, це включає виклик іншого компонента ШІ, або можливо це включає сотні чи навіть тисячі рядків коду в решті вашої системи. Це повністю залежить від вас.

Іноді виклик функції є кінцем операції, але якщо результати представляють інформацію в ланцюжку міркувань, яку має продовжити ШІ, тоді ваш код програми повинен вставити результати виконання в транскрипт чату і дозволити ШІ продовжити обробку.

Наприклад, ось оголошення функції Raix, що використовується AccountManager від Olympia для спілкування з нашими клієнтами як частина Інтелектуальної Оркестрації Робочих Процесів для обслуговування клієнтів.

```
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
```

Можливо, не одразу зрозуміло, що тут відбувається, тому я поясню детальніше.

1. Клас AccountManager визначає багато функцій, пов'язаних з керуванням обліковим записом. Він може змінювати ваш тарифний план, додавати та видаляти учасників команди та виконувати інші дії.
2. Його інструкції верхнього рівня вказують AccountManager, що він повинен повідомляти власника облікового запису про результати запиту на зміну облікового запису, використовуючи функцію notify_account_owner.
3. Стисле визначення функції включає:

- назву
- опис
- параметри message: { type: "string" }
- блок для виконання при виклику функції

Після оновлення транскрипту результатами виконання функціонального блоку, знову викликається метод chat_completion. Цей метод відповідає за відправлення оновленого транскрипту розмови назад до моделі ШІ для подальшої обробки. Ми називаємо цей процес *циклом розмови*.

Коли модель ШІ отримує новий запит на завершення чату з оновленим транскриптом, вона має доступ до результатів попередньо виконаної функції. Вона може проаналізувати ці результати, включити їх у свій процес прийняття рішень і генерувати наступну відповідь або дію на основі накопиченого контексту розмови. Вона може вибрати виконання додаткових функцій на основі оновленого контексту або може генерувати остаточну відповідь на початковий запит, якщо визначить, що додаткові виклики функцій не потрібні.

Необов'язкове продовження початкового запиту

Коли ви відправляєте результати інструменту назад до BMM і продовжуєте обробку початкового запиту, ШІ використовує ці результати для виклику

додаткових функцій або генерування остаточної текстової відповіді.

Деякі моделі, такі як Command-R від Cohere, можуть цитувати конкретні інструменти, які вони використовували у своїх відповідях, забезпечуючи додаткову прозорість і відстежуваність.

Залежно від моделі, що використовується, результати виклику функції будуть знаходитися в повідомленнях транскрипту, які мають свою спеціальну роль, або відображатимуться в іншому синтаксисі. Але важливою частиною є те, щоб ці дані були в транскрипті, щоб ШІ могла врахувати їх при прийнятті рішення про подальші дії.

Поширеною (і потенційно дорогою) помилкою є забування додати результати функції до транскрипту перед продовженням чату. В результаті ШІ отримає запит практично такий самий, як і перед першим викликом функції. Іншими словами, з точки зору ШІ, вона ще не викликала функцію. Тому вона викликає її знову. І знову. І знову, нескінченно, поки ви не перервете її. Сподіваємося, що ваш контекст був не надто великим, а модель не надто дорогою!

Найкращі практики використання інструментів

Щоб максимально ефективно використовувати інструменти, розгляньте наступні найкращі практики.

Описові визначення

Надавайте чіткі та описові назви та описи для кожного інструменту та його вхідних параметрів. Це допомагає BMM краще зрозуміти призначення та можливості кожного інструменту.

> З власного досвіду можу сказати, що поширена мудрість про те, що “вибір назв - це складно” застосовується і тут; я бачив кардинально різні результати від BMM лише через зміну назв функцій або формулювання описів. Іноді видалення описів *покращує* продуктивність.

Обробка результатів інструментів

При передачі результатів інструментів назад до BMM, переконайтеся, що вони добре структуровані та вичерпні. Використовуйте змістовні ключі та значення для представлення виводу кожного інструменту. Експериментуйте з різними форматами та дивіться, який працює найкраще, від JSON до звичайного тексту.

Інтерпретатор результатів вирішує цю проблему, використовуючи ШІ для аналізу результатів та надання зрозумілих для людини пояснень, резюме або ключових висновків.

Обробка помилок

Впроваджуйте надійні механізми обробки помилок для випадків, коли BMM може генерувати недійсні або непідтримувані вхідні параметри для викликів інструментів. Граційно обробляйте та відновлюйтеся після будь-яких помилок, які можуть виникнути під час виконання інструменту.

Однією надзвичайно хорошою якістю ШІ є те, що вона розуміє повідомлення про помилки! Це означає, що якщо ви працюєте в швидкому та недбалому режимі, ви можете просто перехопити будь-які винятки, згенеровані при реалізації інструменту, і передати їх назад до ШІ, щоб вона знала, що сталося!

Наприклад, ось спрощена версія реалізації пошуку Google в Olympia:

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

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

Пошукові запити Google в Olympia - це двоетапний процес. Спочатку ви виконуєте пошук, потім узагальнюєте результати. Якщо виникає помилка, незалежно від її типу, повідомлення про виняток упаковується і відправляється назад до ШІ. Ця техніка є основою практично всіх шаблонів *Інтелектуальної обробки помилок*.

Наприклад, припустімо, що виклик API GoogleSearch завершується невдачею через виняток 503 Service Unavailable. Це передається вгору до верхньорівневого обробника помилок, і опис помилки відправляється назад до ШІ як результат виклику функції. Замість того, щоб просто показати користувачеві порожній екран або технічну помилку, ШІ каже щось на кшталт “Вибачте, але наразі я не можу отримати доступ до можливостей пошуку Google. Я можу спробувати пізніше, якщо бажаєте.”

Це може здаватися просто хитрим трюком, але розгляньмо інший тип помилки, коли ШІ викликає зовнішній API і має прямий контроль над параметрами, які передаються до API. Можливо, він припустився помилки у формуванні цих параметрів? За умови, що повідомлення про помилку від зовнішнього API достатньо детальне, передача повідомлення про помилку назад до ШІ, що викликав функцію, означає, що він може переглянути ці параметри і спробувати знову. Автоматично. Незалежно від того, якою була помилка.

Тепер подумайте, що знадобилося б для відтворення такої надійної обробки

помилок у *звичайному* коді. Це практично неможливо.

Ітеративне вдосконалення

Якщо BMM не рекомендує відповідні інструменти або генерує неоптимальні відповіді, виконуйте ітерації визначень інструментів, описів та вхідних параметрів. Постійно вдосконалюйте та покращуйте налаштування інструментів на основі спостережуваної поведінки та бажаних результатів.

1. Почніть з простих визначень інструментів: Почніть з визначення інструментів з чіткими та лаконічними назвами, описами та вхідними параметрами. Спочатку уникайте надмірного ускладнення налаштувань інструментів і зосередьтеся на основній функціональності. Наприклад, якщо ви хочете зберегти результати аналізу тональності, почніть з базового визначення, як-от:

```
{
  "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. Тестуйте та спостерігайте: Після того, як ви створили початкові визначення інструментів, протестуйте їх різними запитами та спостерігайте, як BMM взаємодіє з інструментом. Зверніть увагу на якість та доречність

згенерованих відповідей. Якщо ВММ генерує неоптимальні відповіді, час уточнити визначення інструментів.

3. Уточніть описи: Якщо ВММ неправильно розуміє призначення інструменту, спробуйте уточнити опис інструменту. Надайте більше контексту, прикладів або роз'яснень, щоб спрямувати ВММ до ефективного використання інструменту. Наприклад, ви можете оновити опис інструменту аналізу тональності, щоб конкретніше визначити *емоційний тон* тексту, що аналізується:

```
{
  "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. Коригуйте вхідні параметри: Якщо ВММ генерує недійсні або недоречні вхідні параметри для інструмента, розгляньте можливість коригування визначень параметрів. Додайте більш конкретні обмеження, правила валідації або приклади, щоб прояснити очікуваний формат введення.
5. Виконуйте ітерації на основі відгуків: Постійно відстежуйте ефективність ваших інструментів і збирайте відгуки від користувачів та зацікавлених сторін. Використовуйте ці відгуки для визначення областей для вдосконалення та внесення послідовних покращень до визначень інструментів. Наприклад, якщо користувачі повідомляють, що аналіз не справляється належним чином із сарказмом, ви можете додати примітку в опис:

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

Шляхом ітеративного вдосконалення визначень ваших інструментів на основі спостережуваної поведінки та зворотного зв'язку, ви можете поступово покращувати продуктивність та ефективність вашого AI-керованого застосунку. Пам'ятайте, що визначення інструментів повинні бути чіткими, лаконічними та зосередженими на конкретному завданні. Регулярно тестуйте та перевіряйте взаємодію інструментів, щоб переконатися, що вони відповідають бажаним результатам.

Компонування та Об'єднання Інструментів

Одним із найпотужніших аспектів використання інструментів, про який досі лише згадувалося, є можливість компонувати та об'єднувати декілька інструментів для виконання складних завдань. Ретельно розробляючи визначення ваших інструментів та їх формати введення/виведення, ви можете створювати багаторазові будівельні блоки, які можна комбінувати різними способами.

Розглянемо приклад, де ви створюєте конвеєр аналізу даних для вашого AI-керованого застосунку. У вас можуть бути такі інструменти:

1. **DataRetrieval**: Інструмент, який отримує дані з бази даних або API на основі визначених критеріїв.
2. **DataProcessing**: Інструмент, який виконує обчислення, перетворення або агрегацію отриманих даних.

3. **DataVisualization**: Інструмент, який представляє оброблені дані у зручному для користувача форматі, наприклад, у вигляді діаграм або графіків.

Об'єднуючи ці інструменти разом, ви можете створити потужний робочий процес, який отримує відповідні дані, обробляє їх та представляє результати у змістовний спосіб. Ось як може виглядати робочий процес використання інструментів:

1. BMM отримує запит користувача щодо аналізу даних про продажі для певної категорії продуктів.
2. BMM вибирає інструмент DataRetrieval і генерує відповідні вхідні параметри для отримання релевантних даних про продажі з бази даних.
3. Отримані дані "передаються" до інструменту DataProcessing, який обчислює такі показники, як загальний дохід, середня ціна продажу та темп зростання.
4. Оброблені дані потім обробляються інструментом DataVisualization, який створює візуально привабливу діаграму або графік для представлення аналітичних даних, передаючи URL-адресу діаграми назад до BMM.
5. Нарешті, BMM генерує відформатовану відповідь на запит користувача, використовуючи markdown, включаючи візуалізовані дані та надаючи підсумок ключових висновків.

Компонуючи ці інструменти разом, ви можете створити безперебійний процес аналізу даних, який можна легко інтегрувати у ваш застосунок. Краса цього підходу полягає в тому, що кожен інструмент можна розробляти та тестувати незалежно, а потім комбінувати різними способами для вирішення різних проблем.

Щоб забезпечити плавне компонування та об'єднання інструментів, важливо визначити чіткі формати введення та виведення для кожного інструмента.

Наприклад, інструмент DataRetrieval може приймати такі параметри, як деталі підключення до бази даних, назва таблиці та умови запиту, і повертати набір результатів як структурований JSON-об'єкт. Інструмент DataProcessing потім може очікувати цей JSON-об'єкт як вхідні дані і створювати перетворений JSON-об'єкт як вихідні дані. Стандартизуючи потік даних між інструментами, ви можете забезпечити сумісність та можливість повторного використання.

Під час проектування вашої екосистеми інструментів, подумайте про те, як різні інструменти можна комбінувати для вирішення поширених випадків використання у вашому застосунку. Розгляньте можливість створення високорівневих інструментів, які інкапсулюють поширені робочі процеси або бізнес-логіку, полегшуючи ВММ їх вибір та ефективне використання.

Пам'ятайте, що сила використання інструментів полягає у гнучкості та модульності, яку воно забезпечує. Розбиваючи складні завдання на менші, багаторазові інструменти, ви можете створити надійний та адаптивний AI-керований застосунок, здатний вирішувати широкий спектр завдань.

Майбутні Напрямки

Оскільки галузь розробки AI-керованих застосунків розвивається, ми можемо очікувати подальших досягнень у можливостях використання інструментів. Деякі потенційні майбутні напрямки включають:

1. **Багатокрокове Використання Інструментів**: ВММ можуть визначати, скільки разів їм потрібно використовувати інструменти для генерації задовільної відповіді. Це може включати кілька раундів вибору та виконання інструментів на основі проміжних результатів.
2. **Попередньо Визначені Інструменти**: AI-платформи можуть надавати набір попередньо визначених інструментів, які розробники можуть

використовувати “з коробки”, таких як інтерпретатори Python, інструменти веб-пошуку або загальні службові функції.

3. **Безперебійна Інтеграція**: Оскільки використання інструментів стає все більш поширеним, ми можемо очікувати кращої інтеграції між AI-платформами та популярними фреймворками розробки, що полегшить розробникам включення використання інструментів у їхні застосунки.

Використання інструментів - це потужна техніка, яка дозволяє розробникам використовувати повний потенціал BMM у AI-керованих застосунках. Підключаючи BMM до зовнішніх інструментів та ресурсів, ви можете створювати більш динамічні, інтелектуальні та контекстно-орієнтовані системи, які можуть адаптуватися до потреб користувачів та надавати цінні аналітичні дані та дії.

Хоча використання інструментів відкриває величезні можливості, важливо усвідомлювати потенційні виклики та міркування. Одним із ключових аспектів є управління складністю взаємодії інструментів та забезпечення стабільності й надійності всієї системи. Вам потрібно обробляти сценарії, коли виклики інструментів можуть завершитися невдачею, повернути неочікувані результати або мати наслідки для продуктивності. Крім того, слід враховувати заходи безпеки та контролю доступу для запобігання несанкціонованому або зловмисному використанню інструментів. Належна обробка помилок, ведення журналів та механізми моніторингу є критично важливими для підтримки цілісності та продуктивності вашого AI-керованого застосунку.

Досліджуючи можливості використання інструментів у власних проєктах, пам’ятайте про необхідність починати з чітких цілей, розробляти добре структуровані визначення інструментів та виконувати ітерації на основі зворотного зв’язку та результатів. За правильного підходу та мислення, використання інструментів може відкрити нові рівні інновацій та цінності у ваших застосунках на основі ШІ

Потокова обробка

Потокова передача даних через HTTP, також відома як події, що надсилаються сервером (SSE), – це механізм, за допомогою якого сервер безперервно надсилає дані клієнту в міру їх доступності, без необхідності явного запиту з боку клієнта. Оскільки відповідь ШІ генерується поступово, має сенс забезпечити чуйний користувацький досвід, відображаючи вивід ШІ в процесі його генерації. І насправді всі API провайдерів ШІ, які я знаю, пропонують потокові відповіді як опцію у своїх кінцевих точках завершення.

Причина, чому цей розділ з'являється тут у книзі, одразу після Використання інструментів, полягає в тому, наскільки потужним може бути поєднання використання інструментів із живими відповідями ШІ користувачам. Це дозволяє створювати динамічні та інтерактивні взаємодії, де ШІ може обробляти користувацький ввід, використовувати різні інструменти та функції на свій розсуд, а потім надавати відповіді в реальному часі.

Для досягнення такої безперебійної взаємодії вам потрібно написати обробники потоку, які можуть відправляти як виклики функцій інструментів, ініційовані ШІ, так і звичайний текстовий вивід кінцевому користувачу. Необхідність зациклювання після обробки функції інструменту додає цікавий виклик до завдання.

Реалізація ReplyStream

Щоб продемонструвати, як можна реалізувати потокову обробку, цей розділ глибоко розгляне спрощену версію класу ReplyStream, який використовується в Olympia. Екземпляри цього класу можуть передаватися як параметр stream у бібліотеках клієнтів ШІ, таких як ruby-openai та openrouter

Ось як я використовую ReplyStream в PromptSubscriber Olympia, який прослуховує через Wisper створення нових повідомлень користувача.

```
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...
```

Окрім посилання context на підписника промпту, який його створив, клас ReplyStream також має змінні екземпляра для зберігання буфера отриманих даних та масиви для відстеження назв функцій і аргументів, що викликаються під час обробки потоку.

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

  delegate :bot_message, :dispatch, to: :context

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

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

  # ...
end
```

Метод initialize налаштовує початковий стан екземпляра ReplyStream, ініціалізуючи буфер, контекст та інші змінні.

Метод call є головною точкою входу для обробки потокових даних. Він приймає частину даних (chunk, представлену як хеш) та необов'язковий параметр bytesize, який у нашому прикладі не використовується. Всередині цього методу клас використовує зіставлення зі зразком для обробки різних сценаріїв на основі структури отриманої частини даних.

Виклик deep_symbolize_keys для частини даних допомагає зробити зіставлення зі зразком більш елегантним, дозволяючи нам працювати з символами замість рядків.

```
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
```

Перший шаблон, який ми зіставляємо, це виклик інструменту разом із відповідним іменем функції. Якщо ми його виявляємо, то поміщаємо його в масив f_name. Ми зберігаємо імена функцій в індексованому масиві, оскільки модель здатна до паралельного виклику функцій, надсилаючи для виконання більше однієї функції одночасно.

> Паралельний виклик функцій - це здатність моделі ШІ виконувати кілька викликів функцій разом, дозволяючи ефектам та результатам цих викликів функцій вирішуватися паралельно. Це особливо корисно, якщо функції виконуються довго, і зменшує кількість циклів обміну з API, що в свою чергу може значно зменшити витрати токенів.

Далі нам потрібно зіставити аргументи, що відповідають викликам функцій.

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

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

Подібно до того, як ми обробляли імена функцій, ми розміщуємо аргументи в індексованому масиві.

Далі ми шукаємо звичайні повідомлення для користувача, які надходитимуть із сервера по одному токену за раз і призначатимуться змінній new_content. Також нам потрібно стежити за finish_reason. Він буде nil, доки не надійде остання частина вихідної послідовності.

```
  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
```

Важливо зазначити, що ми додаємо вираз зіставлення зі зразком для обробки повідомлень про помилки, надісланих провайдером AI-моделі. У локальних

середовищах розробки, ми викидаємо виняток, але у продакшені ми записуємо помилку в журнал і завершуємо виконання.

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

Останній блок else оператора case виконається, якщо жоден з попередніх шаблонів не збігся. Це просто захисний механізм, щоб ми дізналися, якщо модель ШІ почне надсилати нам нерозпізнані фрагменти.

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

Метод send_to_client відповідає за надсилання буферизованого вмісту клієнту. Він перевіряє, що буфер не порожній, оновлює вміст повідомлення бота, відображає повідомлення бота та зберігає вміст у базі даних для забезпечення збереження даних.

```ruby
def send_to_client
  # no need to process pure whitespace
  return if buffer.join.squish.blank?

  # set the buffer content on the bot message
  content = buffer.join
  bot_message.content = content

  # save to database so that we never lose data
  # even if the stream doesn't terminate correctly
  bot_message.update_column(:content, content)

  # update content via websocket
  ConversationRenderer.update(bot_message)
end
```

Метод finalize викликається після завершення потокової обробки. Він відправляє виклики функцій, якщо такі були отримані під час потоку, оновлює повідомлення бота фінальним вмістом та іншою відповідною інформацією, та скидає історію викликів функцій

```ruby
def finalize
  if f_name.any?
    f_name.each_with_index do |name, index|
      # takes care of calling the function wherever it's implemented
      dispatch(name:, arguments: JSON.parse(f_arguments[index]))
    end

    # reset the function call history
    f_name.clear
    f_arguments.clear
  else
    content = buffer.join.presence
    bot_message.update!(content:, complete: true)
    ConversationRenderer.update(bot_message)
  end
end
```

Якщо модель вирішує викликати функцію, вам потрібно виконати “диспетчеризацію” цього виклику функції (назву та аргументи) таким чином,

щоб він виконався, а повідомлення function_call та function_result були додані до протоколу розмови

З мого досвіду, краще обробляти створення повідомлень про функції в одному місці вашої кодової бази, замість того, щоб покладатися на реалізації інструментів. Це не лише чистіше рішення, але й має дуже важливу практичну причину: якщо ШІ-модель викликає функцію, і не бачить результуючих повідомлень про виклик та результат у протоколі під час наступної ітерації, *вона викличе ту саму функцію знову*. Потенційно нескінченно. Пам'ятайте, що ШІ повністю позбавлений стану, тому якщо ви не відобразите ці виклики функцій назад до нього, для нього вони не відбулися.

```
# 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
```

Очищення історії викликів функцій після їх виконання так само важливе, як і забезпечення того, щоб виклик та результати потрапили до вашого протоколу, щоб ви не продовжували викликати одні й ті ж функції знову і знову при кожному проході циклу.

"Цикл розмови"

Я постійно згадую про цикли, але якщо ви новачок у викликах функцій, можливо, не очевидно, *чому* нам потрібен цикл. Причина в тому, що коли ШІ "просить"

вас виконати інструментальні функції від його імені, він припиняє відповідати. Саме ви повинні виконати ці функції, зібрати результати, додати їх до протоколу, а потім знову надіслати початковий запит, щоб отримати новий набір викликів функцій або результатів для користувача.

У класі PromptSubscriber ми використовуємо метод prompt з модуля PromptDeclarations для визначення поведінки циклу розмови. Параметр until встановлено як -> { bot_message.complete? }, що означає, що цикл продовжуватиметься, доки bot_message не буде позначено як завершений.

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

Але коли bot_message позначається як завершений? Якщо ви забули, зверніться до рядка 13 методу finalize.

Давайте розглянемо всю логіку потокової обробки.

1. PromptSubscriber отримує нове повідомлення користувача через метод message_created, який викликається системою публікації/підписки Wisper щоразу, коли кінцевий користувач створює новий запит.
2. Метод класу prompt декларативно визначає поведінку логіки завершення чату для PromptSubscriber. Модель ШІ виконає завершення чату з вмістом повідомлення користувача, новим екземпляром ReplyStream як параметром потоку та вказаною умовою циклу.
3. Модель ШІ обробляє запит і починає генерувати відповідь. Під час потокової передачі відповіді метод call екземпляра ReplyStream викликається для кожного фрагмента даних.
4. Якщо модель ШІ вирішує викликати інструментальну функцію, назва функції та аргументи витягуються з фрагмента та зберігаються у масивах f_name та f_arguments відповідно.

5. Якщо модель ШІ генерує контент для користувача, він буферизується та надсилається клієнту через метод send_to_client.
6. Після завершення потокової обробки викликається метод finalize. Якщо під час потоку були викликані будь-які інструментальні функції, вони відправляються за допомогою методу dispatch класу PromptSubscriber.
7. Метод dispatch додає повідомлення function_call до стенограми розмови, виконує відповідну інструментальну функцію та додає повідомлення function_result до стенограми з результатом виклику функції.
8. Після відправлення інструментальних функцій історія викликів функцій очищається, щоб запобігти дублюванню викликів функцій у наступних циклах.
9. Якщо інструментальні функції не викликалися, метод finalize оновлює bot_message остаточним вмістом, позначає його як завершений та надсилає оновлене повідомлення клієнту.
10. Оцінюється умова циклу -> { bot_message.complete? }. Якщо bot_message не позначено як завершений, цикл продовжується, і початковий запит подається знову з оновленою стенограмою розмови.
11. Кроки 3-10 повторюються, доки bot_message не буде позначено як завершений, що вказує на те, що модель ШІ закінчила генерувати свою відповідь і подальші інструментальні функції не потрібно виконувати.

Реалізуючи цей цикл розмови, ви даєте можливість моделі ШІ взаємодіяти з додатком, виконувати інструментальні функції за потреби та генерувати відповіді для користувача, доки розмова не досягне природного завершення.

Поєднання потокової обробки та циклу розмови дозволяє створювати динамічні та інтерактивні можливості на базі ШІ, де модель ШІ може обробляти введення користувача, використовувати різні інструменти та функції, а також надавати відповіді в реальному часі на основі контексту розмови, що розвивається.

Автоматичне продовження

Важливо знати про обмеження виводу ШІ. Більшість моделей мають максимальну кількість токенів, які вони можуть згенерувати в одній відповіді, що визначається параметром max_tokens. Якщо модель ШІ досягає цього ліміту під час генерації відповіді, вона раптово зупиниться і вкаже, що вивід було обрізано.

У потоковій відповіді від API платформи ШІ ви можете виявити цю ситуацію, перевіривши змінну finish_reason у фрагменті. Якщо finish_reason встановлено на ”length” (або інше ключове значення, специфічне для моделі), це означає, що модель досягла свого максимального ліміту токенів під час генерації, і вивід було обрізано.

Один із способів елегантно обробити цей сценарій та забезпечити безперебійний користувацький досвід - це реалізувати механізм автоматичного продовження у вашій логіці потокової обробки. Додавши зіставлення шаблонів для причин завершення, пов’язаних з довжиною, ви можете вибрати зациклення та автоматичне продовження виводу з того місця, де він зупинився.

Ось навмисно спрощений приклад того, як можна модифікувати метод call у класі ReplyStream для підтримки автоматичного продовження:

```
LENGTH_STOPS = %w[length MAX_TOKENS]

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

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

      buffer << new_content.to_s

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

    # ...
  end
end

private

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

У цій модифікованій версії, коли finish_reason вказує на обрізаний вивід, замість завершення потоку, ми додаємо пару повідомлень до транскрипту без фіналізації, переміщуємо оригінальне користувацьке повідомлення-відповідь у “низ” транскрипту, оновлюючи його атрибут created_at, і потім дозволяємо циклу продовжитися, щоб ШІ міг продовжити генерацію з місця, де зупинився.

Пам’ятайте, що кінцева точка завершення ШІ не має стану. Вона “знає” лише

те, що ви повідомляєте їй через транскрипт. У цьому випадку, спосіб, яким ми повідомляємо ШІ про те, що він був обрізаний - це додавання “невидимих” (для кінцевого користувача) повідомлень до транскрипту. Проте пам’ятайте, що це навмисно спрощений приклад. Реальна реалізація потребуватиме додаткового управління транскриптом, щоб гарантувати, що ми не витрачаємо токени даремно та/або не заплутуємо ШІ дубльованими повідомленнями асистента в транскрипті.

Реальна реалізація автоматичного продовження також повинна мати так звану “логіку автоматичного переривання”, щоб запобігти безконтрольному зациклюванню. Причина в тому, що за певних типів користувацьких запитів та низьких налаштувань max_tokens, ШІ може нескінченно продовжувати генерувати користувацький вивід.

> Майте на увазі, що кожен цикл вимагає окремого запиту, і кожен запит знову споживає весь ваш транскрипт. Ви повинні серйозно зважити компроміси між користувацьким досвідом та використанням API, приймаючи рішення про впровадження автоматичного продовження у вашому додатку. Автоматичне продовження може бути особливо небезпечно дорогим, особливо при використанні преміальних комерційних моделей.

Висновок

Потокова обробка є критичним аспектом створення додатків на базі ШІ, які поєднують використання інструментів з живими відповідями ШІ. Ефективно обробляючи потокові дані з API платформ ШІ, ви можете забезпечити безперебійний та інтерактивний користувацький досвід, обробляти великі відповіді, оптимізувати використання ресурсів та коректно обробляти помилки.

Наданий клас Conversation::ReplyStream демонструє, як потокова обробка може бути реалізована в Ruby-додатку з використанням зіставлення зі зразком та подієво-орієнтованої архітектури. Розуміючи та використовуючи методи потокової обробки, ви можете розкрити повний потенціал інтеграції ШІ у ваших додатках та забезпечити потужний та захоплюючий користувацький досвід.

Самовідновлювані дані

Самовідновлювані дані - це потужний підхід до забезпечення цілісності, узгодженості та якості даних у додатках завдяки можливостям великих мовних моделей (LLM). Ця категорія патернів зосереджена на ідеї використання ШІ для автоматичного виявлення, діагностики та виправлення аномалій, невідповідностей чи помилок у даних, тим самим зменшуючи навантаження на розробників та підтримуючи високий рівень надійності даних.

В основі патернів самовідновлюваних даних лежить розуміння того, що дані є життєвою силою будь-якого додатку, і забезпечення їх точності та цілісності є критично важливим для належного функціонування та користувацького досвіду додатку. Однак управління та підтримка якості даних може бути складним і трудомістким завданням, особливо коли додатки зростають за розміром та складністю. Саме тут на допомогу приходить потужність ШІ.

У патернах самовідновлюваних даних ШІ-працівники використовуються для постійного моніторингу та аналізу даних вашого додатку. Ці моделі здатні розуміти та інтерпретувати шаблони, зв'язки та аномалії в даних. Використовуючи свої можливості обробки та розуміння природної мови, вони можуть виявляти потенційні проблеми чи невідповідності в даних і вживати відповідних заходів для їх виправлення.

Процес самовідновлення даних зазвичай включає кілька ключових кроків:

1. **Моніторинг даних**: ШІ-працівники постійно відстежують потоки даних, бази даних або системи зберігання додатку, шукаючи будь-які ознаки аномалій, невідповідностей чи помилок. Альтернативно, ви можете активувати компонент ШІ у відповідь на виняткову ситуацію.
2. **Виявлення аномалій**: Коли проблему виявлено, ШІ-працівник детально аналізує дані, щоб визначити конкретну природу та масштаб проблеми. Це може включати виявлення відсутніх значень, невідповідних форматів або даних, що порушують попередньо визначені правила чи обмеження.
3. **Діагностика та виправлення**: Після виявлення проблеми ШІ-працівник використовує свої знання та розуміння предметної області даних для визначення відповідного курсу дій. Це може включати автоматичне виправлення даних, заповнення відсутніх значень або позначення проблеми для втручання людини, якщо це необхідно.
4. **Постійне навчання (опціонально, залежно від випадку використання)**: Коли ваш ШІ-працівник стикається з різними проблемами даних та вирішує їх, він може виводити інформацію про те, що сталося і як він відреагував. Ці метадані можуть бути використані в процесах навчання, що дозволяє вам (і, можливо, базовій моделі через точне налаштування) ставати все ефективнішими з часом у виявленні та вирішенні аномалій даних.

Автоматично виявляючи та виправляючи проблеми з даними, ви можете

забезпечити роботу вашого додатку з високоякісними, надійними даними. Це зменшує ризик того, що помилки, невідповідності або помилки, пов'язані з даними, вплинуть на функціональність додатку або користувацький досвід.

Коли у вас є ШІ-працівники, які займаються моніторингом та виправленням даних, ви можете зосередити свої зусилля на інших критичних аспектах додатку. Це економить час та ресурси, які в іншому випадку були б витрачені на ручне очищення та обслуговування даних. Фактично, в міру зростання розміру та складності ваших додатків, ручне управління якістю даних стає все складнішим. Патерни "Самовідновлюваних даних" ефективно масштабуються, використовуючи потужність ШІ для обробки великих обсягів даних та виявлення проблем у реальному часі.

Завдяки своїй природі, моделі ШІ можуть адаптуватися до змін у шаблонах даних, схемах або вимогах з часом з мінімальним наглядом або без нього. Доки їхні директиви забезпечують адекватне керівництво, особливо щодо очікуваних результатів, ваш додаток може еволюціонувати та обробляти нові сценарії даних без необхідності значного ручного втручання або змін у коді.

Патерни самовідновлюваних даних добре узгоджуються з іншими категоріями патернів, які ми обговорювали, такими як "Множина працівників". Можливість самовідновлення даних можна розглядати як спеціалізований вид працівника, який зосереджується саме на забезпеченні якості та цілісності даних. Такий працівник працює поряд з іншими ШІ-працівниками, кожен з яких робить свій внесок у різні аспекти функціональності додатку.

Впровадження патернів самовідновлюваних даних на практиці вимагає ретельного проектування та інтеграції моделей ШІ в архітектуру додатку. Через ризики втрати та пошкодження даних, ви повинні визначити чіткі правила використання цієї техніки. Також слід враховувати такі фактори, як продуктивність, масштабованість та безпека даних.

Практичний приклад: Виправлення пошкодженого JSON

Один з найпрактичніших і найпростіших для пояснення способів використання самовідновлюваних даних - це виправлення пошкодженого JSON.

Цю техніку можна застосувати до поширеної проблеми роботи з недосконалими або неузгодженими даними, створеними LLM, такими як пошкоджений JSON, і вона надає підхід для автоматичного виявлення та виправлення цих проблем.

В Olympia я регулярно стикаюся з ситуаціями, коли BMM генерують дані JSON, які не є повністю валідними. Це може статися з різних причин, наприклад, коли BMM додає коментарі до або після самого коду JSON, або вносить синтаксичні помилки, такі як пропущені коми чи неекрановані подвійні лапки. Ці проблеми можуть призвести до помилок парсингу та спричинити порушення функціональності програми.

Щоб вирішити цю проблему, я реалізував практичне рішення у вигляді класу JsonFixer. Цей клас втілює патерн “Самовідновлюваних Даних”, приймаючи пошкоджений JSON як вхідні дані та використовуючи BMM для його виправлення, зберігаючи при цьому якомога більше інформації та початкового наміру.

```ruby
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
```

Зверніть увагу, як JsonFixer використовує Ventriloquist для керування відповідями ШІ.

Процес самовідновлення JSON-даних працює наступним чином:

1. **Генерація JSON**: ВММ використовується для генерації JSON-даних на основі певних запитів або вимог. Однак, через природу ВММ, згенерований JSON не завжди може бути повністю коректним. JSON-парсер, звісно, видасть помилку ParserError, якщо ви надасте йому некоректний JSON.

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

Зверніть увагу, що повідомлення про помилку також передається до виклику JSONFixer, тому йому не потрібно повністю припускати, що не так з даними, особливо враховуючи, що парсер часто точно вказує на проблему.

2. **Виправлення на основі ВММ**: Клас JSONFixer надсилає пошкоджений JSON назад до ВММ, разом із конкретною інструкцією щодо виправлення JSON, зберігаючи при цьому початкову інформацію та намір настільки, наскільки це можливо. ВММ, навчена на величезних обсягах даних та з розумінням синтаксису JSON, намагається виправити помилки та згенерувати коректний JSON-рядок. Обмеження відповіді використовується для обмеження виводу ВММ, і ми обираємо Mixtral 8x7B як модель ШІ, оскільки вона особливо добре підходить для такого типу завдань.
3. **Перевірка та інтеграція**: Виправлений JSON-рядок, повернутий ВММ, парситься самим класом JSONFixer, оскільки він викликав chat_completion(json: true). Якщо виправлений JSON проходить валідацію, він інтегрується назад у робочий процес програми, дозволяючи їй безперебійно продовжувати обробку даних. Пошкоджений JSON було "вилікувано".

Хоча я писав і переписував власну реалізацію JSONFixer багато разів, я сумніваюся, що загальний час, витрачений на всі ці версії, перевищує годину-дві.

Зверніть увагу, що збереження наміру є ключовим елементом будь-якого патерну самовідновлюваних даних. Процес виправлення на основі ВММ спрямований на збереження оригінальної інформації та наміру згенерованого

JSON настільки, наскільки це можливо. Це гарантує, що виправлений JSON зберігає своє семантичне значення і може ефективно використовуватися в контексті програми.

Ця практична реалізація підходу "Самовідновлюваних даних" в Olympia чітко демонструє, як ШІ, зокрема BMM, можна використовувати для вирішення реальних проблем з даними. Вона показує силу поєднання традиційних методів програмування з можливостями ШІ для створення надійних та ефективних програм.

Закон Постела та патерн "Самовідновлюваних даних"

"Самовідновлювані дані", як показано на прикладі класу JSONFixer, добре узгоджується з принципом, відомим як Закон Постела, який також називають Принципом надійності. Закон Постела стверджує:

"Будьте консервативні в тому, що ви робите, будьте ліберальні в тому, що приймаєте від інших."

Цей принцип, спочатку сформульований Джоном Постелом, піонером раннього Інтернету, підкреслює важливість створення систем, які толерантні до різноманітних або навіть дещо неправильних вхідних даних, зберігаючи при цьому строге дотримання визначених протоколів при надсиланні вихідних даних.

У контексті "Самовідновлюваних даних" клас JSONFixer втілює Закон Постела, будучи ліберальним у прийнятті пошкоджених або недосконалих JSON-даних, згенерованих BMM. Він не відхиляє і не виходить з ладу негайно при зіткненні з JSON, який строго не відповідає очікуваному формату. Натомість він застосовує толерантний підхід і намагається виправити JSON,

використовуючи можливості BMM.

Будучи ліберальним у прийнятті недосконалого JSON, клас JSONFixer демонструє надійність та гнучкість. Він визнає, що дані в реальному світі часто надходять у різних формах і не завжди можуть відповідати строгим специфікаціям. Грацiйно обробляючи та виправляючи ці відхилення, клас забезпечує безперебійну роботу програми навіть за наявності недосконалих даних.

З іншого боку, клас JSONFixer також дотримується консервативного аспекту Закону Постела щодо вихідних даних. Після виправлення JSON за допомогою BMM, клас перевіряє виправлений JSON, щоб переконатися, що він строго відповідає очікуваному формату. Він підтримує цілісність та правильність даних перед передачею їх іншим частинам програми. Цей консервативний підхід гарантує, що вихідні дані класу JSONFixer є надійними та послідовними, сприяючи взаємодії та запобігаючи поширенню помилок.

Цікаві факти про Джона Постела:

- Джон Постел (1943-1998) був американським вченим-комп'ютерником, який відіграв вирішальну роль у розвитку Інтернету. Його називали "Богом Інтернету" за значний внесок у розробку базових протоколів та стандартів.
- Постел був редактором серії документів Request for Comments (RFC), яка є серією технічних та організаційних заміток про Інтернет. Він був автором або співавтором понад 200 RFC, включаючи фундаментальні протоколи, такі як TCP, IP та SMTP.
- Окрім технічного внеску, Постел був відомий своїм скромним та колаборативним підходом. Він вірив у важливість досягнення консенсусу та спільної роботи для створення надійної та взаємодіючої мережі.
- Постел працював директором Відділу комп'ютерних мереж в Інституті

інформаційних наук (ISI) Університету Південної Каліфорнії (USC) з 1977 року до своєї передчасної смерті в 1998 році.
- На знак визнання його величезного внеску, Постелу посмертно було присуджено престижну Премію Тюрінга в 1998 році, яку часто називають "Нобелівською премією в галузі обчислювальної техніки."

Клас JSONFixer сприяє надійності, гнучкості та взаємодії, які були основними цінностями, яких Постел дотримувався протягом своєї кар'єри. Створюючи системи, які толерантні до недосконалостей, зберігаючи при цьому суворе дотримання протоколів, ми можемо створювати програми, які є більш стійкими та адаптивними перед реальними викликами.

Міркування та протипоказання

Застосовність підходів самовідновлюваних даних повністю залежить від типу даних, з якими працює ваша програма. Є причина, чому ви можете не захотіти просто застосувати монкіпатч до JSON.parse, щоб автоматично виправляти *всі помилки розбору JSON* у вашій програмі: не всі помилки можна чи варто автоматично виправляти.

Самовідновлення особливо проблематичне, коли воно пов'язане з нормативними вимогами або вимогами відповідності щодо обробки даних. Деякі галузі, такі як охорона здоров'я та фінанси, мають такі суворі правила щодо цілісності даних та можливості аудиту, що виконання будь-якого виправлення даних за принципом "чорної скриньки" без належного нагляду чи журналювання може порушити ці правила. Важливо переконатися, що всі методи самовідновлення даних, які ви розробляєте, відповідають застосовним правовим та нормативним рамкам.

Застосування методів самовідновлення даних, особливо тих, що включають моделі ШІ, також може мати значний вплив на продуктивність програми та

використання ресурсів. Обробка великих обсягів даних через моделі ШІ для виявлення та виправлення помилок може бути обчислювально інтенсивною. Важливо оцінити компроміси між перевагами самовідновлення даних та пов'язаними витратами на продуктивність і ресурси.

Тим не менш, давайте заглибимось у фактори, які впливають на рішення про те, коли і де застосовувати цей потужний підхід.

Критичність даних

При розгляді застосування методів самовідновлення даних важливо оцінити критичність даних, що обробляються. Рівень критичності відноситься до важливості та чутливості даних у контексті вашої програми та її бізнес-домену.

У деяких випадках автоматичне виправлення помилок даних може бути недоречним, особливо якщо дані є високочутливими або мають юридичні наслідки. Наприклад, розгляньте такі сценарії:

1. **Фінансові транзакції**: У фінансових програмах, таких як банківські системи або торгові платформи, точність даних має найвищу важливість. Навіть незначні помилки у фінансових даних можуть мати серйозні наслідки, такі як неправильні залишки на рахунках, неправильно спрямовані кошти або помилкові торгові рішення. У таких випадках автоматизовані виправлення без ретельної перевірки та аудиту можуть створювати неприйнятні ризики.
2. **Медичні записи**: Програми охорони здоров'я працюють з високочутливими та конфіденційними даними пацієнтів. Неточності в медичних записах можуть мати серйозні наслідки для безпеки пацієнтів та рішень щодо лікування. Автоматична модифікація медичних даних без належного нагляду та валідації кваліфікованими медичними працівниками може порушувати нормативні вимоги та ставити під загрозу благополуччя пацієнтів.

3. **Юридичні документи**: Програми, що обробляють юридичні документи, такі як контракти, угоди або судові документи, вимагають суворої точності та цілісності. Навіть незначні помилки в юридичних даних можуть мати значні правові наслідки. Автоматизовані виправлення в цій області можуть бути недоречними, оскільки дані часто вимагають ручного перегляду та перевірки юридичними експертами для забезпечення їх дійсності та можливості виконання.

У цих критичних сценаріях даних ризики, пов'язані з автоматизованими виправленнями, часто переважають потенційні переваги. Наслідки внесення помилок або неправильної модифікації даних можуть бути серйозними, призводячи до фінансових втрат, юридичної відповідальності або навіть шкоди людям.

При роботі з високо критичними даними важливо надавати пріоритет процесам ручної перевірки та валідації. Людський нагляд та експертиза мають вирішальне значення для забезпечення точності та цілісності даних. Автоматизовані методи самовідновлення все ще можуть використовуватися для позначення потенційних помилок або невідповідностей, але остаточне рішення щодо виправлень повинно включати людське судження та схвалення.

Однак важливо зазначити, що не всі дані в програмі можуть мати однаковий рівень критичності. В межах однієї програми можуть бути підмножини даних, які є менш чутливими або мають менший вплив у разі виникнення помилок. У таких випадках методи самовідновлення даних можуть вибірково застосовуватися до цих конкретних підмножин даних, тоді як критичні дані залишаються предметом ручної перевірки.

Ключовим є ретельна оцінка критичності кожної категорії даних у вашій програмі та визначення чітких рекомендацій і процесів для обробки виправлень на основі пов'язаних ризиків та наслідків. Розрізняючи критичні (тобто бухгалтерські книги, медичні записи) та некритичні дані (тобто поштові адреси,

попередження про ресурси), ви можете знайти баланс між використанням переваг методів самовідновлення даних там, де це доречно, та підтриманням суворого контролю та нагляду там, де це необхідно.

Зрештою, рішення про застосування методів самовідновлення даних до критичних даних повинно прийматися в консультації з експертами в предметній області, юридичними консультантами та іншими відповідними зацікавленими сторонами. Важливо враховувати конкретні вимоги, правила та ризики, пов'язані з даними вашої програми, та відповідно узгоджувати стратегії виправлення даних.

Серйозність помилок

При застосуванні методів самовідновлення даних важливо оцінити серйозність та вплив помилок у даних. Не всі помилки однакові, і відповідні дії можуть відрізнятися залежно від серйозності проблеми.

Незначні невідповідності або проблеми форматування можуть підходити для автоматичного виправлення. Наприклад, працівник з самовідновлення даних, призначений для виправлення пошкодженого JSON, може обробляти відсутні коми або неекрановані подвійні лапки без значної зміни значення або структури даних. Такі типи помилок часто легко виправити і мають мінімальний вплив на загальну цілісність даних.

Проте серйозніші помилки, які принципово змінюють значення чи цілісність даних, можуть вимагати іншого підходу. У таких випадках автоматизованих виправлень може бути недостатньо, і може знадобитися втручання людини для забезпечення точності та достовірності даних.

Саме тут вступає в дію концепція використання самого ШІ для визначення критичності помилок. Використовуючи можливості моделей ШІ, ми можемо розробляти самовідновлювальні обробники даних, які не лише виправляють

помилки, але й оцінюють їхню критичність та приймають обґрунтовані рішення щодо того, як з ними працювати.

Наприклад, розглянемо самовідновлювальний обробник даних, відповідальний за виправлення невідповідностей у даних, що надходять до бази даних клієнтів. Обробник можна налаштувати на аналіз даних та виявлення потенційних помилок, таких як відсутня або суперечлива інформація. Однак замість автоматичного виправлення всіх помилок, обробник може бути оснащений додатковими викликами інструментів, які дозволяють позначати серйозні помилки для перевірки людиною.

Ось приклад того, як це можна реалізувати:

```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" }
```

```
27
28      self.stop = ["```"]
29
30      chat_completion(json: true).then do |result|
31        return if result.blank?
32
33        customer.update(result)
34      end
35    end
36  end
```

У цьому прикладі працівник CustomerDataHealer розроблений для виявлення та виправлення невідповідностей у даних клієнтів. Знову ж таки, ми використовуємо Обмеження відповідей та Вентрилоквіст для отримання структурованого виводу. Важливо зазначити, що системна директива працівника містить інструкції щодо використання функції flag_for_review у випадку виявлення серйозних помилок.

Коли працівник обробляє дані клієнтів, він аналізує дані та намагається виправити будь-які невідповідності. Якщо працівник визначає, що помилки є серйозними і потребують втручання людини, він може використати інструмент flag_for_review для позначення даних та надання причини такого позначення.

Метод chat_completion викликається з параметром json: true для розбору виправлених даних клієнта як JSON. Немає можливості зациклювання після виклику функції, тому результат буде порожнім, якщо було викликано flag_for_review. В іншому випадку, дані клієнта оновлюються переглянутими та потенційно виправленими даними.

Включаючи оцінку серйозності помилок та можливість позначати дані для перегляду людиною, працівник з самовідновлення даних стає більш розумним та адаптивним. Він може автоматично обробляти незначні помилки, одночасно ескалуючи серйозні помилки до експертів-людей для ручного втручання.

Конкретні критерії визначення серйозності помилок можуть бути визначені в

директиві працівника на основі доменних знань та бізнес-вимог. При оцінці серйозності можуть враховуватися такі фактори, як вплив на цілісність даних, потенціал втрати або пошкодження даних, та наслідки неправильних даних.

Використовуючи ШІ для оцінки серйозності помилок та надаючи можливості для втручання людини, методи самовідновлення даних можуть досягти балансу між автоматизацією та підтримкою точності даних. Цей підхід забезпечує ефективне виправлення незначних помилок, тоді як серйозні помилки отримують необхідну увагу та експертизу від людей-рецензентів.

Складність домену

При розгляді застосування методів самовідновлення даних важливо оцінити складність домену даних та правил, що регулюють їх структуру та взаємозв'язки. Складність домену може суттєво вплинути на ефективність та здійсненність підходів автоматизованого виправлення даних.

Методи самовідновлення даних добре працюють, коли дані відповідають чітко визначеним шаблонам та обмеженням. У доменах, де структура даних відносно проста, а зв'язки між елементами даних прямолінійні, автоматизовані виправлення можуть застосовуватися з високим ступенем впевненості. Наприклад, виправлення проблем форматування або забезпечення базових обмежень типів даних часто може ефективно оброблятися працівниками самовідновлення даних.

Однак, зі збільшенням складності домену даних, зростають і виклики, пов'язані з автоматизованим виправленням даних. У доменах зі складною бізнес-логікою, складними відносинами між сутностями даних або специфічними для домену правилами та винятками, методи самовідновлення даних не завжди можуть охопити всі нюанси та можуть призвести до непередбачуваних наслідків.

Розглянемо приклад складного домену: система фінансової торгівлі. У цьому домені дані включають різні фінансові інструменти, ринкові дані, правила

торгівлі та регуляторні вимоги. Зв'язки між різними елементами даних можуть бути складними, а правила, що регулюють достовірність та узгодженість даних, можуть бути високоспецифічними для домену.

У такому складному домені працівник самовідновлення даних, якому доручено виправляти невідповідності в торгових даних, повинен мати глибоке розуміння специфічних для домену правил та обмежень. Він повинен враховувати такі фактори, як ринкові регуляції, торгові ліміти, розрахунки ризиків та процедури розрахунків. Автоматизовані виправлення в цьому контексті не завжди можуть охопити всю складність домену і можуть ненавмисно внести помилки або порушити специфічні для домену правила.

Для вирішення проблем складності домену методи самовідновлення даних можуть бути покращені шляхом включення специфічних для домену знань та правил у моделі ШІ та працівників. Цього можна досягти за допомогою таких методів:

1. **Доменно-специфічне навчання**: Моделі ШІ, що використовуються для самовідновлення даних, можуть бути спрямовані або навіть точно налаштовані на специфічних для домену наборах даних, які охоплюють складності та правила конкретного домену. Показуючи моделям репрезентативні дані та сценарії, вони можуть вивчити шаблони, обмеження та винятки, специфічні для домену.
2. **Правило-орієнтовані обмеження**: Працівники самовідновлення даних можуть бути доповнені явними правило-орієнтованими обмеженнями, які кодують специфічні для домену знання. Ці правила можуть бути визначені експертами домену та інтегровані в процес виправлення даних. Моделі ШІ можуть потім використовувати ці правила для керування своїми рішеннями та забезпечення відповідності специфічним для домену вимогам.
3. **Співпраця з експертами домену**: У складних доменах критично важливо залучати експертів домену до проектування та розробки методів

самовідновлення даних. Експерти домену можуть надати цінні уявлення про складності даних, бізнес-правила та потенційні граничні випадки. Їхні знання можуть бути включені в моделі ШІ та працівників для покращення точності та надійності автоматизованих виправлень даних, використовуючи патерни Людина в циклі.

4. **Поступовий та ітеративний підхід**: При роботі зі складними доменами часто корисно прийняти поступовий та ітеративний підхід до самовідновлення даних. Замість того, щоб намагатися автоматизувати виправлення для всього домену одразу, зосередьтеся на конкретних піддоменах або категоріях даних, де правила та обмеження добре зрозумілі. Поступово розширюйте сферу застосування методів самовідновлення в міру зростання розуміння домену та доведення ефективності методів.

Враховуючи складність предметної області даних та включаючи предметно-орієнтовані знання в методи самовідновлення даних, можна досягти балансу між автоматизацією та точністю. Важливо розуміти, що самовідновлення даних не є універсальним рішенням, і підхід має бути адаптований до конкретних вимог та викликів кожної предметної області.

У складних предметних областях найефективнішим може бути гібридний підхід, який поєднує методи самовідновлення даних із експертними знаннями людей та наглядом. Автоматизовані виправлення можуть обробляти рутинні та чітко визначені випадки, тоді як складні сценарії чи винятки можуть позначатися для перегляду та втручання людини. Такий спільний підхід забезпечує реалізацію переваг автоматизації при збереженні необхідного контролю та точності в складних предметних областях даних.

Пояснюваність та прозорість

Пояснюваність стосується здатності розуміти та інтерпретувати логіку рішень, прийнятих моделями штучного інтелекту, тоді як прозорість передбачає

забезпечення чіткої видимості процесу виправлення даних.

У багатьох контекстах модифікації даних повинні бути придатними для аудиту та обґрунтованими. Зацікавлені сторони, включаючи бізнес-користувачів, аудиторів та регуляторні органи, можуть вимагати пояснень щодо того, чому були зроблені певні виправлення даних і як моделі ШІ прийшли до цих рішень. Це особливо важливо в областях, де точність та цілісність даних мають значні наслідки, таких як фінанси, охорона здоров'я та юридичні питання.

Для забезпечення потреби в пояснюваності та прозорості, методи самовідновлення даних повинні включати механізми, які надають розуміння процесу прийняття рішень моделями ШІ. Цього можна досягти різними підходами:

1. **Ланцюжок міркувань**: Запит моделі пояснити своє мислення "вголос" перед внесенням змін до даних може полегшити розуміння процесу прийняття рішень і може генерувати зрозумілі для людини пояснення зроблених виправлень. Компромісом є трохи більша складність у відокремленні пояснення від структурованого виведення даних, що можна вирішити через...
2. **Генерація пояснень**: Працівники з самовідновлення даних можуть бути оснащені здатністю генерувати зрозумілі для людини пояснення виправлень, які вони роблять. Цього можна досягти, попросивши модель виводити свій процес прийняття рішень як легко зрозумілі пояснення, *інтегровані в самі дані*. Наприклад, працівник з самовідновлення даних може генерувати звіт, який висвітлює конкретні невідповідності даних, які він виявив, застосовані виправлення та обґрунтування цих виправлень.
3. **Важливість ознак**: Моделі ШІ можуть бути проінструктовані інформацією про важливість різних ознак або атрибутів у процесі виправлення даних як частина їхніх директив. Ці директиви, в свою чергу, можуть бути доступні для зацікавлених сторін-людей. Визначаючи ключові

фактори, що впливають на рішення моделі, зацікавлені сторони можуть отримати розуміння логіки, що стоїть за виправленнями, та оцінити їх обґрунтованість.

4. **Журналювання та аудит**: Впровадження комплексних механізмів журналювання та аудиту є критично важливим для підтримки прозорості в процесі самовідновлення даних. Кожне виправлення даних, зроблене моделями ШІ, повинно бути зареєстроване, включаючи оригінальні дані, виправлені дані та конкретні вжиті дії. Цей контрольний журнал дозволяє проводити ретроспективний аналіз і надає чіткий запис змін, внесених до даних.
5. **Підхід із залученням людини**: Включення підходу із залученням людини може покращити пояснюваність та прозорість методів самовідновлення даних. Залучаючи експертів-людей до перегляду та валідації виправлень, згенерованих ШІ, організації можуть забезпечити відповідність виправлень предметним знанням та бізнес-вимогам. Нагляд людини додає додатковий рівень відповідальності та дозволяє виявити будь-які потенційні упередження чи помилки в моделях ШІ.
6. **Постійний моніторинг та оцінка**: Регулярний моніторинг та оцінка роботи методів самовідновлення даних є важливими для підтримки прозорості та довіри. Оцінюючи точність та ефективність моделей ШІ з часом, організації можуть виявити будь-які відхилення чи аномалії та вжити коригувальних заходів. Постійний моніторинг допомагає забезпечити надійність процесу самовідновлення даних та його відповідність бажаним результатам.

Пояснюваність та прозорість є критичними міркуваннями при впровадженні методів самовідновлення даних. Надаючи чіткі пояснення для виправлень даних, підтримуючи комплексні контрольні журнали та залучаючи нагляд людини, організації можуть побудувати довіру до процесу самовідновлення

даних та забезпечити, що модифікації, внесені до даних, є обґрунтованими та відповідають бізнес-цілям.

Важливо досягти балансу між перевагами автоматизації та потребою в прозорості. Хоча методи самовідновлення даних можуть значно покращити якість даних та ефективність, це не повинно відбуватися за рахунок втрати видимості та контролю над процесом виправлення даних. Проектуючи працівників з самовідновлення даних з урахуванням пояснюваності та прозорості, організації можуть використовувати потужність ШІ, зберігаючи необхідний рівень відповідальності та довіри до даних.

Непередбачені наслідки

Хоча методи самовідновлення даних спрямовані на покращення якості та узгодженості даних, важливо усвідомлювати можливість непередбачених наслідків. Автоматизовані виправлення, якщо вони не ретельно спроектовані та не контролюються, можуть ненавмисно змінити значення або контекст даних, що призведе до проблем у подальшому.

Одним із основних ризиків самовідновлення даних є введення упередженості або помилок у процес виправлення даних. Моделі ШІ, як і будь-яка інша програмна система, можуть бути схильні до упереджень, присутніх у навчальних даних або введених через дизайн алгоритмів. Якщо ці упередження не виявлені та не пом'якшені, вони можуть поширюватися через процес самовідновлення даних і призвести до викривлених або неправильних модифікацій даних.

Наприклад, розглянемо обробник самовідновлюваних даних, якому доручено виправляти невідповідності в демографічних даних клієнтів. Якщо модель ШІ засвоїла упередження з історичних даних, такі як пов'язування певних професій або рівнів доходу з конкретними статями чи етнічними групами, вона може робити неправильні припущення та змінювати дані таким чином, що посилює

ці упередження. Це може призвести до неточних профілів клієнтів, помилкових бізнес-рішень та потенційно дискримінаційних результатів.

Іншим потенційним ненавмисним наслідком є втрата цінної інформації або контексту під час процесу виправлення даних. Методи самовідновлення даних часто зосереджуються на стандартизації та нормалізації даних для забезпечення узгодженості. Проте в деяких випадках початкові дані можуть містити нюанси, винятки або контекстну інформацію, важливу для розуміння повної картини. Автоматизовані виправлення, які сліпо застосовують стандартизацію, можуть ненавмисно видалити або приховати цю цінну інформацію.

Наприклад, уявіть обробник самовідновлюваних даних, відповідальний за виправлення невідповідностей у медичних записах. Якщо обробник зустрічає історію хвороби пацієнта з рідкісним захворюванням або незвичайним планом лікування, він може спробувати нормалізувати дані, щоб вони відповідали більш поширеному шаблону. Однак при цьому можуть бути втрачені конкретні деталі та контекст, які є критично важливими для точного відображення унікальної ситуації пацієнта. Така втрата інформації може мати серйозні наслідки для догляду за пацієнтом та прийняття медичних рішень.

Щоб зменшити ризики ненавмисних наслідків, важливо застосовувати проактивний підхід при розробці та впровадженні методів самовідновлення даних:

1. **Ретельне тестування та валідація**: Перед розгортанням обробників самовідновлюваних даних у виробничому середовищі критично важливо ретельно протестувати та перевірити їхню поведінку на різноманітних сценаріях. Це включає тестування на репрезентативних наборах даних, що охоплюють різні граничні випадки, винятки та потенційні упередження. Ретельне тестування допомагає виявити та усунути будь-які ненавмисні наслідки до того, як вони вплинуть на реальні дані.

2. **Постійний моніторинг та оцінка**: Впровадження механізмів постійного моніторингу та оцінки є важливим для виявлення та пом'якшення ненавмисних наслідків з часом. Регулярний перегляд результатів процесів самовідновлення даних, аналіз впливу на залежні системи та прийняття рішень, а також збір відгуків від зацікавлених сторін можуть допомогти виявити будь-які негативні ефекти та вчасно вжити коригувальних заходів. Якщо у вашій організації є операційні інформаційні панелі, ймовірно, варто додати чітко видимі метрики, пов'язані з автоматизованими змінами даних. Додавання сигналів тривоги, пов'язаних із великими відхиленнями від нормальної активності зміни даних, ймовірно, є ще кращою ідеєю!
3. **Людський нагляд та втручання**: Підтримка людського нагляду та можливості втручання в процес самовідновлення даних є критично важливою. Хоча автоматизація може значно підвищити ефективність, важливо мати експертів-людей для перегляду та перевірки виправлень, зроблених моделями ШІ, особливо в критичних або чутливих областях. Людське судження та експертні знання предметної області можуть допомогти виявити та усунути будь-які ненавмисні наслідки, які можуть виникнути.

4. **Пояснюваний ШІ (XAI) та прозорість**: Як обговорювалося в попередньому підрозділі, включення методів пояснюваного ШІ та забезпечення прозорості в процесі самовідновлення даних може допомогти пом'якшити ненавмисні наслідки. Надаючи чіткі пояснення для виправлень даних та підтримуючи комплексні журнали аудиту, організації можуть краще розуміти та відстежувати логіку змін, внесених моделями ШІ.
5. **Поступовий та ітеративний підхід**: Прийняття поступового та ітеративного підходу до самовідновлення даних може допомогти мінімізувати ризик ненавмисних наслідків. Замість того, щоб застосовувати автоматизовані виправлення до всього набору даних одразу, почніть з підмножини даних і поступово розширюйте масштаб,

коли методи доведуть свою ефективність та надійність. Це дозволяє здійснювати ретельний моніторинг та коригування в процесі, зменшуючи вплив будь-яких ненавмисних наслідків.

6. **Співпраця та зворотний зв'язок**: Залучення зацікавлених сторін з різних областей та заохочення співпраці і зворотного зв'язку протягом усього процесу самовідновлення даних може допомогти виявити та усунути ненавмисні наслідки. Регулярне отримання вхідних даних від експертів предметної області, споживачів даних та кінцевих користувачів може надати цінну інформацію про реальний вплив виправлень даних та висвітлити будь-які проблеми, які могли бути пропущені.

Проактивно вирішуючи ризик ненавмисних наслідків та впроваджуючи відповідні захисні заходи, організації можуть використовувати переваги методів самовідновлення даних, мінімізуючи при цьому потенційні негативні ефекти. Важливо підходити до самовідновлення даних як до ітеративного та спільного процесу, постійно моніторити, оцінювати та вдосконалювати методи, щоб забезпечити їх відповідність бажаним результатам та підтримувати цілісність і надійність даних.

При розгляді використання шаблонів самовідновлення даних важливо ретельно оцінити ці фактори та зважити переваги порівняно з потенційними ризиками та обмеженнями. У деяких випадках гібридний підхід, що поєднує автоматизовані виправлення з людським наглядом та втручанням, може бути найбільш доцільним рішенням.

Також варто зазначити, що методи самовідновлення даних не слід розглядати як заміну надійним механізмам валідації даних, санітизації вхідних даних та обробки помилок. Ці фундаментальні практики залишаються критично важливими для забезпечення цілісності та безпеки даних. Самовідновлення

даних слід розглядати як додатковий підхід, який може доповнювати та покращувати ці існуючі заходи.

Зрештою, рішення про використання шаблонів самовідновлення даних залежить від конкретних вимог, обмежень та пріоритетів вашого додатка. Ретельно розглянувши викладені вище міркування та узгодивши їх з цілями та архітектурою вашого додатка, ви зможете приймати обґрунтовані рішення щодо того, коли і як ефективно використовувати методи самовідновлення даних.

Контекстне Генерування Контенту

Патерни контекстного генерування контенту використовують потужність великих мовних моделей (ВММ) для створення динамічного та контекстно-специфічного вмісту в додатках. Ця категорія патернів визнає важливість надання персоналізованого та релевантного контенту користувачам на основі їхніх конкретних потреб, уподобань та навіть попередніх і поточних взаємодій з додатком.

> У контексті цього підходу “контент” стосується як основного вмісту (тобто блог-постів, статей тощо), так і мета-контенту, такого як рекомендації до основного вмісту.

Патерни контекстного генерування контенту можуть відігравати вирішальну роль у підвищенні рівня залучення користувачів, наданні персоналізованого досвіду та автоматизації завдань зі створення контенту як для вас, так і для ваших користувачів. Використовуючи патерни, які ми описуємо в цьому розділі, ви можете створювати додатки, що динамічно генерують контент, адаптуючись до контексту та вхідних даних у реальному часі.

Патерни працюють шляхом інтеграції ВММ у вихідні дані додатка, починаючи від користувацького інтерфейсу (іноді називається “chrome”), до електронних листів та інших форм сповіщень, а також будь-яких конвеєрів генерування контенту.

Коли користувач взаємодіє з додатком або ініціює певний запит на контент, додаток фіксує відповідний контекст, такий як уподобання користувача, попередні взаємодії або конкретні запити. Ця контекстуальна інформація потім подається у ВММ разом із необхідними шаблонами чи рекомендаціями та використовується для створення текстового виводу, який в іншому випадку довелося б або жорстко кодувати, зберігати в базі даних, або генерувати алгоритмічно.

Контент, згенерований ВММ, може набувати різних форм, таких як персоналізовані рекомендації, динамічні описи продуктів, персоналізовані відповіді на електронні листи або навіть цілі статті чи блог-пости. Одним із найрадикальніших застосувань такого контенту, який я започаткував більше року тому, є динамічне генерування елементів інтерфейсу, таких як мітки форм, підказки та інші види пояснювального тексту.

Персоналізація

Однією з ключових переваг патернів контекстного генерування контенту є можливість надавати користувачам високоперсоналізований досвід. Генеруючи

контент на основі користувацького контексту, ці патерни дозволяють додаткам адаптувати вміст до інтересів, уподобань та взаємодій окремих користувачів.

Персоналізація виходить за межі простого вставлення імені користувача в загальний контент. Вона передбачає використання багатого контексту, доступного про кожного користувача, для генерування контенту, який резонує з їхніми конкретними потребами та бажаннями. Цей контекст може включати широкий спектр факторів, таких як:

1. **Інформація профілю користувача**: На найзагальнішому рівні застосування цієї техніки, демографічні дані, інтереси, уподобання та інші атрибути профілю можуть використовуватися для генерування контенту, який відповідає походженню та характеристикам користувача.
2. **Поведінкові дані**: Минулі взаємодії користувача з додатком, такі як переглянуті сторінки, натиснуті посилання або придбані продукти, можуть надати цінну інформацію про їхню поведінку та інтереси. Ці дані можуть використовуватися для генерування пропозицій контенту, що відображає їхні шаблони взаємодії та передбачає їхні майбутні потреби.
3. **Контекстуальні фактори**: Поточний контекст користувача, такий як його місцезнаходження, пристрій, час доби чи навіть погода, може впливати на процес генерування контенту. Наприклад, туристичний додаток може мати ШІ-працівника, здатного генерувати персоналізовані рекомендації на основі поточного місцезнаходження користувача та поточних погодних умов.

Використовуючи ці контекстуальні фактори, патерни контекстного генерування контенту дозволяють додаткам надавати контент, який здається створеним спеціально для кожного окремого користувача. Цей рівень персоналізації має кілька значних переваг:

1. **Підвищене залучення**: Персоналізований контент привертає увагу користувачів і утримує їх залученими до додатка. Коли користувачі

відчувають, що контент є релевантним і безпосередньо відповідає їхнім потребам, вони з більшою ймовірністю проводитимуть більше часу, взаємодіючи з додатком та досліджуючи його функції.

2. **Покращене задоволення користувачів**: Персоналізований контент демонструє, що додаток розуміє та дбає про унікальні вимоги користувача. Надаючи контент, який є корисним, інформативним та відповідає їхнім інтересам, додаток може підвищити задоволеність користувачів та побудувати міцніший зв'язок зі своїми користувачами.
3. **Вищі коефіцієнти конверсії**: У контексті електронної комерції або маркетингових додатків персоналізований контент може значно вплинути на коефіцієнти конверсії. Представляючи користувачам продукти, пропозиції або рекомендації, які адаптовані до їхніх уподобань та поведінки, додаток може підвищити ймовірність того, що користувачі виконають бажані дії, такі як здійснення покупки або реєстрація на послугу.

Продуктивність

Патерни контекстного генерування контенту можуть значно підвищити певні види продуктивності шляхом зменшення потреби в ручному генеруванні та редагуванні контенту в творчих процесах. Використовуючи потужність ВММ, ви можете генерувати високоякісний контент у масштабі, економлячи час та зусилля, які ваші творці контенту та розробники в іншому випадку мали б витрачати на виконання рутинної ручної роботи.

Традиційно створювачі контенту мають досліджувати, писати, редагувати та форматувати контент, щоб забезпечити його відповідність вимогам застосунку та очікуванням користувачів. Цей процес може бути трудомістким та ресурсозатратним, особливо коли обсяг контенту зростає.

Проте, з патернами Контекстної генерації контенту процес створення контенту може бути значною мірою автоматизований. ВММ можуть генерувати зв'язний,

граматично правильний та контекстуально релевантний контент на основі наданих підказок та рекомендацій. Ця автоматизація надає декілька переваг для продуктивності:

1. **Зменшення ручної роботи**: Делегуючи завдання генерації контенту ВММ, створювачі контенту можуть зосередитися на завданнях вищого рівня, таких як контентна стратегія, генерація ідей та забезпечення якості. Вони можуть надати необхідний контекст, шаблони та рекомендації для ВММ і дозволити їй займатися безпосередньо генерацією контенту. Це зменшує ручні зусилля, необхідні для написання та редагування, дозволяючи створювачам контенту бути більш продуктивними та ефективними.
2. **Швидше створення контенту**: ВММ можуть генерувати контент набагато швидше, ніж людські автори. За наявності правильних підказок та рекомендацій, ВММ може створювати кілька фрагментів контенту за лічені секунди чи хвилини. Ця швидкість дозволяє застосункам генерувати контент набагато швидше, встигаючи за потребами користувачів та постійно мінливим цифровим ландшафтом.

> Чи призводить швидше створення контенту до ситуації "трагедії спільного", коли інтернет тоне в контенті, який ніхто не читає? На жаль, я підозрюю, що відповідь - так.

3. **Послідовність та якість**: ВММ можуть легко переглядати контент так, щоб він був послідовним за стилем, тоном та якістю. За наявності чітких рекомендацій та прикладів, певні види застосунків (наприклад, редакції новин, PR тощо) можуть забезпечити відповідність свого контенту, створеного людьми, їхньому фірмовому голосу та бажаним стандартам

якості. Ця послідовність зменшує потребу в масштабному редагуванні та переглядах, економлячи час та зусилля в процесі створення контенту.

4. **Ітерація та оптимізація**: Патерни Контекстної генерації контенту уможливлюють швидку ітерацію та оптимізацію контенту. Регулюючи підказки, шаблони чи рекомендації, надані ВММ, ваші застосунки можуть швидко генерувати варіації контенту та тестувати різні підходи в автоматизований спосіб, який ніколи не був можливим у минулому. Цей ітеративний процес дозволяє швидше експериментувати та вдосконалювати контентні стратегії, що з часом призводить до створення ефективнішого та привабливішого контенту. Ця конкретна техніка може стати справжнім проривом для застосунків, таких як електронна комерція, які живуть і вмирають на основі показників відмов та залученості

Важливо зазначити, що хоча патерни Контекстної генерації контенту можуть значно підвищити продуктивність, вони не повністю виключають потребу в людському втручанні. Створювачі контенту та редактори все ще відіграють вирішальну роль у визначенні загальної контентної стратегії, наданні рекомендацій для ВММ та забезпеченні якості й доречності згенерованого контенту.

Автоматизуючи більш повторювані та трудомісткі аспекти створення контенту, патерни Контекстної генерації контенту вивільняють цінний людський час та ресурси, які можна перенаправити на завдання з вищою цінністю. Це підвищення продуктивності дозволяє вам надавати більш персоналізований та привабливий контент користувачам, одночасно оптимізуючи робочі процеси створення контенту.

Швидка ітерація та експериментування

Патерни Контекстної генерації контенту дозволяють швидко ітерувати та експериментувати з різними варіаціями контенту, уможливлюючи швидшу оптимізацію та вдосконалення вашої контентної стратегії. Ви можете генерувати кілька версій контенту за лічені секунди, просто регулюючи контекст, шаблони чи рекомендації, надані моделі.

Ця можливість швидкої ітерації надає кілька ключових переваг:

1. **Тестування та оптимізація**: З можливістю швидко генерувати варіації контенту, ви можете легко тестувати різні підходи та вимірювати їхню ефективність. Наприклад, ви можете генерувати кілька версій опису продукту чи маркетингового повідомлення, кожне з яких адаптоване до конкретного сегменту користувачів чи контексту. Аналізуючи метрики залученості користувачів, такі як показники клікабельності чи конверсії, ви можете визначити найефективніші варіації контенту та відповідно оптимізувати вашу контентну стратегію.

2. **A/B тестування**: Патерни Контекстної генерації контенту уможливлюють безперебійне A/B тестування контенту. Ви можете генерувати дві чи більше варіації контенту та випадковим чином показувати їх різним групам користувачів. Порівнюючи ефективність кожної варіації, ви можете визначити, який контент найкраще резонує з вашою цільовою аудиторією. Цей підхід, заснований на даних, дозволяє приймати обґрунтовані рішення та постійно вдосконалювати ваш контент для максимізації залученості користувачів та досягнення бажаних результатів.
3. **Експерименти з персоналізацією**: Швидка ітерація та експериментування особливо цінні, коли йдеться про персоналізацію. З патернами Контекстної генерації контенту, ви можете швидко генерувати персоналізовані

варіації контенту на основі різних сегментів користувачів, преференцій чи поведінки. Експериментуючи з різними стратегіями персоналізації, ви можете визначити найефективніші підходи для залучення окремих користувачів та надання індивідуалізованого досвіду.

4. **Адаптація до мінливих тенденцій**: Здатність швидко виконувати ітерації та експериментувати дозволяє залишатися гнучкими та адаптуватися до змінних тенденцій і вподобань користувачів. Коли з'являються нові теми, ключові слова чи змінюється поведінка користувачів, ви можете швидко створювати контент, що відповідає цим тенденціям. Постійно експериментуючи та вдосконалюючи свій контент, ви можете залишатися актуальними та зберігати конкурентну перевагу в постійно мінливому цифровому середовищі.
5. **Економічно ефективне експериментування**: Традиційне експериментування з контентом часто вимагає значних витрат часу та ресурсів, оскільки творці контенту повинні вручну розробляти та тестувати різні варіації. Однак із патернами контекстуальної генерації контенту вартість експериментування значно знижується. Великі мовні моделі можуть швидко генерувати варіації контенту в масштабі, дозволяючи досліджувати широкий спектр ідей та підходів без значних витрат.

Щоб максимально ефективно використовувати швидку ітерацію та експериментування, важливо мати чітко визначений фреймворк експериментування. Цей фреймворк повинен включати:

- Чіткі цілі та гіпотези для кожного експерименту
- Відповідні метрики та механізми відстеження для вимірювання ефективності контенту
- Стратегії сегментації та таргетингу для забезпечення відповідних варіацій контенту потрібним користувачам

- Інструменти аналізу та звітності для отримання висновків з експериментальних даних
- Процес включення отриманих знань та оптимізацій у вашу контент-стратегію

Застосовуючи швидку ітерацію та експериментування, ви можете постійно вдосконалювати та оптимізувати свій контент, забезпечуючи його привабливість, актуальність та ефективність у досягненні цілей вашого додатку. Цей гнучкий підхід до створення контенту дозволяє вам залишатися попереду та забезпечувати виняткові користувацькі враження.

Масштабованість та ефективність

З ростом додатків та збільшенням попиту на персоналізований контент, патерни контекстуальної генерації контенту забезпечують ефективне масштабування створення контенту. Великі мовні моделі можуть одночасно генерувати контент для великої кількості користувачів та контекстів без необхідності пропорційного збільшення людських ресурсів. Така масштабованість дозволяє додаткам надавати персоналізований досвід зростаючій базі користувачів без перенавантаження можливостей створення контенту.

Зауважте, що контекстуальну генерацію контенту можна ефективно використовувати для інтернаціоналізації вашого додатку “на льоту”. Насправді, саме це я зробив за допомогою мого гему Instant18n, щоб забезпечити роботу Olympia більш ніж на півдюжині мов, хоча нам менше року.

Локалізація на основі ШІ

Якщо ви дозволите мені похвалитися на мить, я вважаю, що моя бібліотека Instant18n для Rails-додатків є революційним прикладом патерну

“Контекстуальної генерації контенту” в дії, що демонструє трансформаційний потенціал ШІ в розробці додатків. Цей гем використовує потужність великої мовної моделі GPT від OpenAI для революціонізації способу обробки інтернаціоналізації та локалізації в Rails-додатках.

Традиційно, інтернаціоналізація Rails-додатку включає ручне визначення ключів перекладу та надання відповідних перекладів для кожної підтримуваної мови. Цей процес може бути трудомістким, ресурсозатратним та схильним до невідповідностей. Однак з гемом Instant18n парадигма локалізації повністю переосмислюється.

Завдяки інтеграції великої мовної моделі, гем Instant18n дозволяє генерувати переклади на льоту, базуючись на контексті та значенні тексту. Замість того, щоб покладатися на попередньо визначені ключі перекладу та статичні переклади, гем динамічно перекладає текст, використовуючи потужність ШІ. Цей підхід пропонує кілька ключових переваг:

1. **Безшовна локалізація**: З гемом Instant18n розробникам більше не потрібно вручну визначати та підтримувати файли перекладів для кожної підтримуваної мови. Гем автоматично генерує переклади на основі наданого тексту та бажаної цільової мови, роблячи процес локалізації легким та безперешкодним.
2. **Контекстуальна точність**: ШІ можна надати достатньо контексту для розуміння нюансів тексту, що перекладається. Він може враховувати навколишній контекст, ідіоми та культурні посилання для генерації перекладів, які є точними, природними та контекстуально доречними.
3. **Широка підтримка мов**: Гем Instant18n використовує величезні знання та лінгвістичні можливості GPT, забезпечуючи переклади широким спектром мов. Від поширених мов, як іспанська та французька, до більш рідкісних чи вигаданих мов, як клінгонська та ельфійська, гем може впоратися з широким спектром перекладацьких вимог.

4. **Гнучкість та креативність**: Гем виходить за межі традиційних мовних перекладів і дозволяє створювати креативні та нестандартні варіанти локалізації. Розробники можуть перекладати текст різними стилями, діалектами чи навіть вигаданими мовами, відкриваючи нові можливості для унікального користувацького досвіду та привабливого контенту.
5. **Оптимізація продуктивності**: Гем Instant18n включає механізми кешування для покращення продуктивності та зменшення навантаження від повторних перекладів. Перекладений текст кешується, дозволяючи швидко обслуговувати наступні запити на той самий переклад без необхідності надлишкових API-викликів.

Гем Instant18n демонструє потужність патерну "Контекстуальної генерації контенту", використовуючи ШІ для динамічної генерації локалізованого контенту. Він показує, як ШІ можна інтегрувати в основну функціональність Rails-додатку, трансформуючи підхід розробників до інтернаціоналізації та локалізації.

Усуваючи потребу в ручному керуванні перекладами та забезпечуючи динамічний переклад на основі контексту, гем Instant18n значно економить час та зусилля розробників. Це дозволяє їм зосередитися на створенні основних функцій свого застосунку, водночас забезпечуючи безперебійну та точну локалізацію.

Важливість користувацького тестування та зворотного зв'язку

Нарешті, завжди пам'ятайте про важливість користувацького тестування та зворотного зв'язку. Критично важливо перевіряти, що контекстне генерування вмісту відповідає очікуванням користувачів та узгоджується з цілями застосунку. Постійно вдосконалюйте та покращуйте згенерований вміст на

основі користувацьких відгуків та аналітики. Якщо ви генеруєте динамічний вміст у великих масштабах, який неможливо перевірити вручну вами та вашою командою, розгляньте можливість додавання механізмів зворотного зв'язку, які дозволять користувачам повідомляти про дивний чи неправильний вміст, разом із поясненням причини. Цей цінний зворотний зв'язок можна навіть передати ШІ-обробнику, відповідальному за внесення коректив до компонента, який згенерував цей вміст!

Генеративний користувацький інтерфейс

В наші дні увага є настільки цінним ресурсом, що для ефективного залучення користувачів тепер потрібні програмні рішення, які не лише безперешкодні та інтуїтивно зрозумілі, але й максимально персоналізовані відповідно до індивідуальних потреб, уподобань та контексту. Як наслідок, дизайнери та розробники все частіше стикаються з викликом створення користувацьких інтерфейсів, які можуть адаптуватися та відповідати унікальним вимогам кожного користувача *в масштабі.*

Генеративний користувацький інтерфейс (GenUI) - це справді революційний підхід до дизайну користувацького інтерфейсу, який використовує потужність великих мовних моделей (LLMs) для створення високоперсоналізованого та динамічного користувацького досвіду в режимі реального часу. Я хотів

обов'язково дати вам хоча б базове розуміння GenUI в цій книзі, оскільки вважаю, що це одна з найперспективніших можливостей, які наразі існують у сфері проєктування та фреймворків додатків. Я переконаний, що в цій конкретній ніші з'явиться ще десятки успішних комерційних та відкритих проєктів.

У своїй основі GenUI поєднує принципи Контекстної генерації контенту з передовими методами штучного інтелекту для динамічного генерування елементів користувацького інтерфейсу, таких як текст, зображення та макети, на основі глибокого розуміння контексту, уподобань та цілей користувача. GenUI дозволяє дизайнерам та розробникам створювати інтерфейси, які адаптуються та розвиваються у відповідь на взаємодію з користувачем, забезпечуючи рівень персоналізації, який раніше був недосяжним.

GenUI представляє фундаментальну зміну в нашому підході до дизайну користувацького інтерфейсу. Замість проєктування для мас, GenUI дозволяє нам проєктувати для окремої людини. Персоналізований контент та інтерфейси мають потенціал створювати користувацький досвід, який резонує з кожним користувачем на глибшому рівні, підвищуючи залученість, задоволеність та лояльність.

Як передова технологія, перехід до GenUI сповнений концептуальних та практичних викликів. Інтеграція штучного інтелекту в процес проєктування, забезпечення того, щоб згенеровані інтерфейси були не лише персоналізованими, але й зручними, доступними та узгодженими із загальним брендом та користувацьким досвідом - усе це виклики, які роблять GenUI справою для небагатьох, а не для багатьох. Крім того, залучення штучного інтелекту піднімає питання конфіденційності даних, прозорості та навіть етичних наслідків.

Незважаючи на виклики, персоналізований досвід у масштабі має силу повністю трансформувати спосіб нашої взаємодії з цифровими продуктами та послугами. Це відкриває можливості для створення інклюзивних та доступних інтерфейсів,

які відповідають різноманітним потребам користувачів, незалежно від їхніх можливостей, походження чи уподобань.

У цьому розділі ми дослідимо концепцію GenUI, розглянувши деякі визначальні характеристики, ключові переваги та потенційні виклики. Ми починаємо з розгляду найбільш базової та доступної форми GenUI: генерування текстового наповнення для традиційно спроєктованих та реалізованих користувацьких інтерфейсів.

Генерування текстового наповнення для користувацьких інтерфейсів

Текстові елементи, які існують в оформленні вашого додатка, такі як мітки форм, спливаючі підказки та пояснювальний текст, зазвичай жорстко закодовані в шаблонах або компонентах користувацького інтерфейсу, забезпечуючи послідовний, але загальний досвід для всіх користувачів. Використовуючи патерни контекстної генерації контенту, ви можете перетворити ці статичні елементи на динамічні, контекстно-залежні та персоналізовані компоненти.

Персоналізовані форми

Форми є всюдисущою частиною веб- та мобільних додатків, слугуючи основним засобом збору користувацького введення. Однак традиційні форми часто представляють загальний та безособовий досвід, зі стандартними мітками та полями, які не завжди відповідають конкретному контексту або потребам користувача. Користувачі з більшою ймовірністю заповнюють форми, які здаються адаптованими до їхніх потреб та уподобань, що призводить до вищих показників конверсії та задоволеності.

Проте важливо знайти баланс між персоналізацією та послідовністю. Хоча адаптація форм до окремих користувачів може бути корисною, важливо

зберігати певний рівень знайомості та передбачуваності. Користувачі все ще повинні мати можливість легко розпізнавати та орієнтуватися у формах, навіть з персоналізованими елементами.

Ось кілька ідей персоналізованих форм для натхнення:

Контекстні підказки для полів

GenUI може аналізувати попередні взаємодії користувача, уподобання та дані для надання інтелектуальних пропозицій полів як прогнозів. Наприклад, якщо користувач раніше вводив свою адресу доставки, форма може автоматично заповнити відповідні поля їхньою збереженою інформацією. Це не лише економить час, але й демонструє, що додаток розуміє та пам'ятає уподобання користувача.

Зачекайте хвилинку, хіба цю техніку не можна реалізувати без використання ШІ? Звісно можна, але краса впровадження такої функціональності за допомогою ШІ полягає у двох аспектах: 1) наскільки легко це можна реалізувати та 2) наскільки стійким це рішення буде під час змін та еволюції вашого інтерфейсу користувача з часом.

Давайте швидко створимо сервіс для нашої теоретичної системи обробки замовлень, який намагатиметься проактивно заповнювати правильну адресу доставки для користувача.

```ruby
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
```

Цей приклад дуже спрощений, але має працювати у більшості випадків. Ідея полягає в тому, щоб дозволити ШІ робити припущення так само, як це робила б людина. Щоб чітко пояснити, про що я говорю, розглянемо деякі зразки даних:

```
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"}
]
```

Ви помітили закономірність у даних? Я гарантую, що це простий матеріал для ВММ. Щоб продемонструвати це, давайте запитаємо GPT-4, яка найімовірніша адреса доставки для “термометра”.

```
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.
```

Якщо ви думаєте, що використовувати дорогу модель на кшталт GPT-4 для цього завдання - це перебір, ви маєте рацію! Я спробував той самий промпт на Mistral 7B Instruct, і він видав наступну відповідь зі швидкістю 75 токенів за секунду та мізерною вартістю у $0.000218 USD.

```
Based on the given order history, it appears that the customer has received
orders for baby-related items (diapers, wipes, formula, clothes, bath items)
at the address "789 Suburb St, Quietville" and garden-related items (soil,
seeds, watering can, herb garden kit) at the address "123 Country Lane, Rural
Town". The customer has also received books (fiction, non-fiction, literary
classics, mystery-thriller, science fiction, biographies, children's books)
at the address "456 City Apt, Metroville".

Given that the new order is for a "thermometer", which doesn't seem to fit
into any of the categories that the customer has previously ordered, it's
hard to accurately predict the shipping address based solely on the order
history. However, given the 50%-50% split between baby-related and
garden-related items, it could somewhat lean towards the Baby-related items
address ("789 Suburb St, Quietville"). But remember, this is an assumption
and cannot be definitively confirmed without more context or information.
```

Чи варті накладні витрати та вартість цієї техніки того, щоб зробити процес оформлення замовлення більш магічним? Для багатьох онлайн-продавців, безумовно. І судячи з усього, вартість обчислень ШІ буде лише знижуватися, особливо для постачальників послуг хостингу моделей з відкритим кодом, які змагаються за найнижчу ціну.

Використовуйте Шаблон Промпту та StructuredIO разом з Response Fencing для оптимізації такого виду завершення чату.

Адаптивне впорядкування полів

Порядок відображення полів форми може суттєво впливати на користувацький досвід та частоту заповнення. За допомогою GenUI, ви можете динамічно

налаштовувати порядок полів на основі контексту користувача та важливості кожного поля. Наприклад, якщо користувач заповнює реєстраційну форму для фітнес-додатку, форма може надати пріоритет полям, пов'язаним з їхніми фітнес-цілями та вподобаннями, роблячи процес більш релевантним та привабливим.

Персоналізований мікротекст

Інструктивний текст, повідомлення про помилки та інший мікротекст, пов'язаний з формами, також можна персоналізувати за допомогою GenUI. Замість відображення загальних повідомлень про помилки на кшталт "Недійсна електронна адреса", ви можете генерувати більш корисні та контекстуальні повідомлення, такі як "Будь ласка, введіть дійсну електронну адресу для отримання підтвердження замовлення". Такі персоналізовані елементи можуть зробити досвід заповнення форми більш дружнім та менш дратівливим.

Персоналізована валідація

У тому ж дусі, що й Персоналізований мікротекст, ви могли б використовувати ШІ для валідації форми способами, які здаються магічними. Уявіть, що ШІ перевіряє форму профілю користувача, шукаючи потенційні помилки на *семантичному* рівні.

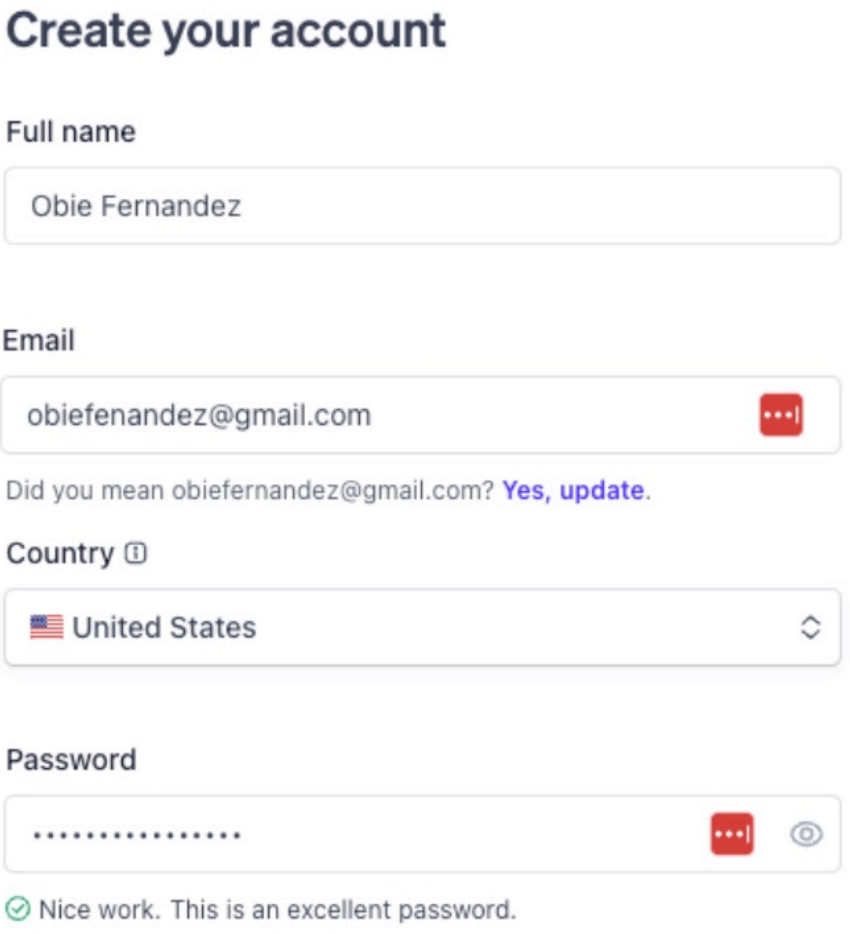


Рисунок 9. Чи можете ви помітити семантичну валідацію, що відбувається?

Поступове розкриття

GenUI може розумно визначати, які поля форми є важливими на основі контексту користувача, та поступово розкривати додаткові поля за потреби. Ця техніка поступового розкриття допомагає зменшити когнітивне навантаження та робить процес заповнення форми більш керованим. Наприклад, якщо користувач

реєструється для базової підписки, форма може спочатку показати лише необхідні поля, а в міру просування користувача або вибору певних опцій можуть динамічно з'являтися додаткові релевантні поля.

Контекстно-залежний пояснювальний текст

Спливаючі підказки часто використовуються для надання додаткової інформації або вказівок користувачам, коли вони наводять курсор або взаємодіють з певними елементами. За допомогою підходу "Контекстної генерації вмісту" ви можете створювати спливаючі підказки, які адаптуються до контексту користувача та надають релевантну інформацію. Наприклад, якщо користувач вивчає складну функцію, спливаюча підказка може запропонувати персоналізовані поради або приклади на основі їхніх попередніх взаємодій або рівня навичок.

Пояснювальний текст, такий як інструкції, описи або повідомлення довідки, може динамічно генеруватися на основі контексту користувача. Замість представлення загальних пояснень, ви можете використовувати LLM для генерації тексту, який адаптований до конкретних потреб або питань користувача. Наприклад, якщо користувач має труднощі з певним кроком процесу, пояснювальний текст може надати персоналізовані вказівки або поради щодо усунення несправностей.

Мікротекст відноситься до невеликих фрагментів тексту, які супроводжують користувачів через ваш додаток, таких як написи на кнопках, повідомлення про помилки або запити на підтвердження. Застосовуючи підхід Контекстної генерації вмісту до мікротексту, ви можете створити адаптивний інтерфейс користувача, який реагує на дії користувача та надає релевантний і корисний текст. Наприклад, якщо користувач збирається виконати критичну дію, запит на підтвердження може бути згенерований динамічно, щоб надати чітке та персоналізоване повідомлення.

Персоналізований пояснювальний текст та спливаючі підказки можуть значно покращити процес онбордингу нових користувачів. Надаючи контекстно-

залежні вказівки та приклади, ви можете допомогти користувачам швидко зрозуміти та орієнтуватися в додатку, зменшуючи криву навчання та збільшуючи рівень прийняття.

Динамічні та контекстно-залежні елементи інтерфейсу також можуть зробити додаток більш інтуїтивним та привабливим. Користувачі з більшою ймовірністю взаємодіятимуть та досліджуватимуть функції, коли супровідний текст адаптований до їхніх конкретних потреб та інтересів.

Досі ми розглядали ідеї щодо вдосконалення наявних парадигм користувацького інтерфейсу за допомогою ШІ, але як щодо того, щоб радикально переосмислити спосіб проєктування та реалізації користувацьких інтерфейсів?

Визначення генеративного інтерфейсу користувача

На відміну від традиційного дизайну інтерфейсу, де дизайнери створюють фіксовані, статичні інтерфейси, GenUI натякає на майбутнє, в якому наше програмне забезпечення матиме гнучкий, персоналізований досвід, що може розвиватися та адаптуватися в реальному часі. Щоразу, коли ми використовуємо розмовний інтерфейс на основі ШІ, ми дозволяємо ШІ адаптуватися до конкретних потреб користувача. GenUI робить крок уперед, застосовуючи такий рівень адаптивності до *візуального* інтерфейсу програмного забезпечення.

Причина, чому сьогодні можливо експериментувати з ідеями GenUI, полягає в тому, що великі мовні моделі вже розуміють програмування, а їхні базові знання включають технології та фреймворки користувацького інтерфейсу. Питання полягає в тому, чи можна використовувати великі мовні моделі для генерації елементів інтерфейсу, таких як текст, зображення, макети та навіть

цілі інтерфейси, які адаптовані під кожного окремого користувача. Модель можна налаштувати так, щоб вона враховувала різні фактори, такі як попередні взаємодії користувача, заявлені вподобання, демографічну інформацію та поточний контекст використання, для створення високоперсоналізованих та релевантних інтерфейсів.

GenUI відрізняється від традиційного дизайну інтерфейсу користувача кількома ключовими аспектами:

1. **Динамічність та адаптивність**: Традиційний дизайн інтерфейсу передбачає створення фіксованих, статичних інтерфейсів, які залишаються однаковими для всіх користувачів. Натомість GenUI дозволяє створювати інтерфейси, які можуть динамічно адаптуватися та змінюватися залежно від потреб користувача та контексту. Це означає, що один і той самий додаток може представляти різні інтерфейси різним користувачам або навіть одному й тому ж користувачеві в різних ситуаціях.
2. **Масштабована персоналізація**: У традиційному дизайні створення персоналізованого досвіду для кожного користувача часто непрактичне через необхідний час та ресурси. З іншого боку, GenUI дозволяє здійснювати персоналізацію в масштабі. Використовуючи ШІ, дизайнери можуть створювати інтерфейси, які автоматично адаптуються до унікальних потреб та вподобань кожного користувача, без необхідності вручну проєктувати та розробляти окремі інтерфейси для кожного сегмента користувачів.
3. **Фокус на результатах**: Традиційний дизайн інтерфейсу часто зосереджується на створенні візуально привабливих та функціональних інтерфейсів. Хоча ці аспекти залишаються важливими в GenUI, основний фокус зміщується на досягнення бажаних результатів користувача. GenUI прагне створювати інтерфейси, оптимізовані під конкретні цілі та завдання кожного користувача, надаючи пріоритет зручності використання та ефективності перед суто естетичними міркуваннями.

4. **Постійне навчання та вдосконалення**: Системи GenUI можуть постійно навчатися та вдосконалюватися з часом на основі взаємодій з користувачами та зворотного зв'язку. Коли користувачі взаємодіють із згенерованими інтерфейсами, моделі ШІ можуть збирати дані про поведінку користувачів, їхні вподобання та результати, використовуючи цю інформацію для вдосконалення та оптимізації майбутніх генерацій інтерфейсу. Цей ітеративний процес навчання дозволяє системам GenUI ставати все ефективнішими у задоволенні потреб користувачів з часом.

Важливо зазначити, що GenUI – це не те саме, що інструменти дизайну з підтримкою ШІ, такі як ті, що надають пропозиції або автоматизують певні завдання дизайну. Хоча ці інструменти можуть бути корисними для оптимізації процесу проєктування, вони все ще покладаються на дизайнерів для прийняття остаточних рішень та створення статичних інтерфейсів. GenUI, натомість, передбачає більш активну роль системи ШІ у фактичній генерації та адаптації інтерфейсів на основі даних користувача та контексту.

GenUI представляє значний зсув у тому, як ми підходимо до дизайну інтерфейсу користувача, відходячи від універсальних рішень і рухаючись до високоперсоналізованого, адаптивного досвіду. Використовуючи потужність ШІ, GenUI має потенціал революціонізувати спосіб нашої взаємодії з цифровими продуктами та послугами, створюючи інтерфейси, які є більш інтуїтивними, захопливими та ефективними для кожного окремого користувача.

Приклад

Щоб проілюструвати концепцію GenUI, розглянемо гіпотетичний фітнес-додаток під назвою "FitAI". Цей додаток має на меті надавати персоналізовані плани тренувань та поради щодо харчування користувачам на основі їхніх індивідуальних цілей, рівня фізичної підготовки та вподобань.

У традиційному підході до дизайну інтерфейсу, FitAI міг би мати фіксований набір екранів та елементів, однакових для всіх користувачів. Однак із GenUI інтерфейс додатку міг би динамічно адаптуватися до унікальних потреб та контексту кожного користувача.

Такий підхід досить складно уявити для реалізації в 2024 році, і він може навіть не мати адекватної рентабельності інвестицій, але це можливо.

Ось як це могло б працювати:

1. **Онбординг**:

    - Замість стандартної анкети, FitAI використовує розмовний ШІ для збору інформації про цілі користувача, поточний рівень фізичної підготовки та вподобання.
    - На основі цієї початкової взаємодії ШІ генерує персоналізований макет інформаційної панелі, виділяючи функції та інформацію, найбільш релевантні для цілей користувача.
    - Сучасні технології ШІ могли б мати у своєму розпорядженні набір екранних компонентів для створення персоналізованої інформаційної панелі.
    - Майбутні технології ШІ могли б взяти на себе роль досвідченого UI-дизайнера і фактично створювати інформаційну панель *з нуля*.

2. **Планувальник тренувань**:

    - Інтерфейс планувальника тренувань адаптується штучним інтелектом відповідно до рівня досвіду користувача та наявного обладнання.
    - Для початківця без обладнання він може показувати прості вправи з власною вагою з детальними інструкціями та відео.
    - Для досвідченого користувача з доступом до спортзалу він може відображати складніші комплекси з меншою кількістю пояснювального контенту.

- Вміст планувальника тренувань не просто фільтрується з великого набору даних. Він може генеруватися *на льоту* на основі бази знань, до якої надходять запити з контекстом, що включає всю відому інформацію про користувача.

3. **Відстеження прогресу**:

   - Інтерфейс відстеження прогресу розвивається на основі цілей користувача та патернів взаємодії.
   - Якщо користувач в першу чергу зосереджений на схудненні, інтерфейс може помітно відображати графік тенденції ваги та статистику спалених калорій.
   - Для користувача, який нарощує м'язи, він може підкреслювати приріст сили та зміни складу тіла.
   - ШІ може адаптувати цю частину програми до фактичного прогресу користувача. Якщо прогрес зупиняється на певний період, додаток може перейти в режим, де він намагається спонукати користувача розкрити причини відставання, щоб їх усунути.

4. **Поради щодо харчування**:

   - Розділ харчування адаптується до дієтичних уподобань та обмежень користувача.
   - Для веганів він може показувати рослинні варіанти страв та джерела білка.
   - Для користувача з непереносимістю глютену він автоматично відфільтровує продукти, що містять глютен, з рекомендацій.
   - Знову ж таки, контент не береться з масивного набору даних про харчування, що застосовується до всіх користувачів, а синтезується з бази знань, яка містить інформацію, що адаптується на основі конкретної ситуації та обмежень користувача.

- Наприклад, рецепти генеруються зі специфікаціями інгредієнтів, які відповідають потребам у калоріях, що постійно змінюються в міру розвитку фізичної форми та показників тіла користувача.

5. **Мотиваційні елементи**:

   - Мотиваційний контент та сповіщення додатку персоналізуються на основі типу особистості користувача та реакції на різні мотиваційні стратегії.
   - Деякі користувачі можуть отримувати підбадьорливі повідомлення, тоді як інші - більш орієнтований на дані відгук.

У цьому прикладі GenUI дозволяє FitAI створювати високо персоналізований досвід для кожного користувача, потенційно підвищуючи залученість, задоволеність та ймовірність досягнення фітнес-цілей. Елементи інтерфейсу, контент і навіть "особистість" додатку адаптуються для найкращого обслуговування потреб та уподобань кожного окремого користувача.

Перехід до проєктування, орієнтованого на результат

GenUI представляє фундаментальний зсув у підході до проєктування інтерфейсів користувача!, переходячи від фокусу на створенні конкретних елементів інтерфейсу до більш цілісного підходу, орієнтованого на результат. Цей зсув має кілька важливих наслідків:

1. **Фокус на цілях користувача**:

   - Дизайнерам потрібно буде глибше замислюватися про цілі користувачів та бажані результати, а не про конкретні компоненти інтерфейсу.

- Акцент буде зроблено на створенні систем, які можуть генерувати інтерфейси, що допомагають користувачам ефективно досягати своїх цілей.
- З'являться нові UI-фреймворки, які нададуть дизайнерам на основі ШІ інструменти, необхідні для генерації користувацького досвіду *на льоту* та *з нуля*, замість базування на попередньо визначених специфікаціях екранів.

2. **Зміна ролі дизайнерів**:

   - Дизайнери перейдуть від створення фіксованих макетів до визначення правил, обмежень та рекомендацій, яких мають дотримуватися системи ШІ при генерації інтерфейсів.
   - Їм потрібно буде розвивати навички в таких областях, як аналіз даних, інженерія промптів для ШІ та системне мислення, щоб ефективно керувати системами GenUI.

3. **Важливість дослідження користувачів**:

   - Дослідження користувачів стає ще більш критичним у контексті GenUI, оскільки дизайнерам потрібно розуміти не лише уподобання користувачів, але й те, як ці уподобання та потреби змінюються в різних контекстах.
   - Постійне тестування користувачів та цикли зворотного зв'язку будуть необхідними для вдосконалення та покращення здатності ШІ генерувати ефективні інтерфейси.

4. **Проєктування для варіативності**:

   - Замість створення єдиного "ідеального" інтерфейсу, дизайнерам потрібно буде враховувати множинні можливі варіації та забезпечувати здатність системи генерувати відповідні інтерфейси для різноманітних потреб користувачів.

- Це включає проєктування для крайових випадків та забезпечення того, щоб згенеровані інтерфейси зберігали зручність використання та доступність у різних конфігураціях.
- Диференціація продукту набуває нових вимірів, що включають різні перспективи щодо психології користувачів та використання унікальних наборів даних і баз знань, недоступних конкурентам.

Виклики та міркування

Хоча GenUI пропонує захоплюючі можливості, він також представляє кілька викликів та міркувань:

1. **Технічні обмеження**:

    - Сучасна технологія ШІ, хоча й розвинена, все ще має обмеження в розумінні складних намірів користувачів та генерації справді контекстно-залежних інтерфейсів.
    - Проблеми з продуктивністю, пов'язані з генерацією елементів інтерфейсу в реальному часі, особливо на менш потужних пристроях.

2. **Вимоги до даних**:

    - Залежно від випадку використання, ефективні системи GenUI можуть потребувати значної кількості користувацьких даних для створення персоналізованих інтерфейсів.
    - Виклики щодо етичного збору автентичних користувацьких даних викликають занепокоєння щодо конфіденційності та безпеки даних, а також потенційних упереджень у даних, що використовуються для навчання моделей GenUI.

3. **Зручність використання та послідовність**:

- Принаймні доки практика не стане широко розповсюдженою, додаток із постійно змінюваними інтерфейсами може призвести до проблем із зручністю використання, оскільки користувачам може бути складно знаходити звичні елементи або ефективно орієнтуватися.
- Критично важливим буде досягнення балансу між персоналізацією та підтримкою послідовного, зрозумілого інтерфейсу.

4. **Надмірна залежність від ШІ**:

   - Існує ризик надмірного делегування дизайнерських рішень системам ШІ, що потенційно може призвести до неоригінальних, проблемних або просто непрацюючих інтерфейсних рішень.
   - Людський нагляд та можливість скасовувати згенеровані ШІ дизайни залишатимуться важливими в найближчому майбутньому.

5. **Проблеми доступності**:

   - Забезпечення доступності динамічно згенерованих інтерфейсів для користувачів з обмеженими можливостями створює абсолютно нові виклики, що викликає занепокоєння, враховуючи низький рівень відповідності вимогам доступності у типових системах.
   - З іншого боку, ШІ-дизайнери можуть бути реалізовані з *вбудованою* увагою до доступності та можливостями створення доступних інтерфейсів на льоту, так само як вони створюють UI для користувачів без обмежень.
   - У будь-якому випадку, системи GenUI повинні розроблятися з урахуванням надійних рекомендацій щодо доступності та процесів тестування.

6. **Довіра користувачів та прозорість**:

- Користувачі можуть відчувати дискомфорт через інтерфейси, які, здається, “занадто багато знають” про них або змінюються незрозумілим для них чином.
- Забезпечення прозорості щодо того, як і чому інтерфейси персоналізуються, буде важливим для побудови довіри користувачів.

Майбутні перспективи та можливості

Майбутнє Генеративного UI (GenUI) має величезний потенціал для революційних змін у способі нашої взаємодії з цифровими продуктами та послугами. По мірі розвитку цієї технології ми можемо очікувати кардинальних змін у тому, як користувацькі інтерфейси проектуються, впроваджуються та сприймаються. Я вважаю, що GenUI - це явище, яке нарешті виведе наше програмне забезпечення у сферу того, що зараз вважається науковою фантастикою.

Однією з найбільш захоплюючих перспектив GenUI є його потенціал покращити доступність у масштабах, що виходять за межі простого забезпечення того, щоб люди з серйозними обмеженими можливостями не були повністю виключені з користування програмним забезпеченням. Автоматично адаптуючи інтерфейси до індивідуальних потреб користувачів, GenUI може зробити цифровий досвід більш інклюзивним, ніж будь-коли раніше. Уявіть інтерфейси, які плавно налаштовуються для відображення більшого тексту для молодших користувачів або користувачів з порушеннями зору, або спрощені макети для людей з когнітивними порушеннями - все це без необхідності ручного налаштування або окремих “доступних” версій додатків.

Можливості персоналізації GenUI, ймовірно, сприятимуть підвищенню залученості користувачів, їхнього задоволення та лояльності в широкому спектрі цифрових продуктів. Оскільки інтерфейси стають більш налаштованими на індивідуальні вподобання та поведінку, користувачі знаходитимуть цифровий

досвід більш інтуїтивним та приємним, що потенційно призведе до глибшої та змістовнішої взаємодії з технологіями.

GenUI також має потенціал трансформувати процес онбордингу нових користувачів. Створюючи інтуїтивний, персоналізований досвід для нових користувачів, який швидко адаптується до рівня експертизи кожного користувача, GenUI може значно зменшити криву навчання, пов'язану з новими додатками. Це може призвести до швидшого впровадження та підвищення впевненості користувачів у дослідженні нових функцій та можливостей.

Іншою захоплюючою можливістю є здатність GenUI підтримувати послідовний користувацький досвід на різних пристроях та платформах, оптимізуючи його для кожного конкретного контексту використання. Це могло б вирішити давню проблему забезпечення узгодженого досвіду в усе більш фрагментованому ландшафті пристроїв, від смартфонів та планшетів до настільних комп'ютерів та новітніх технологій, таких як окуляри доповненої реальності

Орієнтованість GenUI на дані відкриває можливості для швидкої ітерації та вдосконалення дизайну інтерфейсу користувача. Збираючи дані в реальному часі про те, як користувачі взаємодіють зі згенерованими інтерфейсами, дизайнери та розробники можуть отримати безпрецедентне розуміння поведінки та вподобань користувачів. Цей зворотний зв'язок може призвести до постійного вдосконалення дизайну UI, керованого реальними моделями використання, а не припущеннями чи обмеженим користувацьким тестуванням.

Щоб підготуватися до цих змін, дизайнерам потрібно буде розвивати свої навички та мислення. Фокус зміститься від створення фіксованих макетів до розробки комплексних систем дизайну та рекомендацій, які можуть інформувати генерацію інтерфейсів на основі ШІ. Дизайнерам потрібно буде розвивати глибоке розуміння аналізу даних, технологій ШІ та системного мислення для ефективного керування системами GenUI.

Більше того, оскільки GenUI розмиває межі між дизайном і технологіями,

дизайнерам потрібно буде тісніше співпрацювати з розробниками та фахівцями з даних. Цей міждисциплінарний підхід буде критично важливим для створення систем GenUI, які є не лише візуально привабливими та зручними для користувачів, але також технічно надійними та етично обґрунтованими.

Етичні наслідки GenUI також вийдуть на перший план у міру розвитку технології. Дизайнери відіграватимуть вирішальну роль у розробці принципів відповідального використання ШІ в дизайні інтерфейсів, забезпечуючи, щоб персоналізація покращувала користувацький досвід, не порушуючи конфіденційність і не маніпулюючи поведінкою користувачів неетичними способами.

Дивлячись у майбутнє, GenUI відкриває як захоплюючі можливості, так і значні виклики. Ця технологія має потенціал створювати більш інтуїтивний, ефективний та задовільний цифровий досвід для користувачів у всьому світі. Хоча це вимагатиме від дизайнерів адаптації та набуття нових навичок, це також надає безпрецедентну можливість формувати майбутнє взаємодії людини з комп'ютером глибоким і значущим чином. Шлях до повноцінної реалізації систем GenUI, безсумнівно, буде складним, але потенційні переваги з точки зору покращення користувацького досвіду та цифрової доступності роблять це майбутнє вартим прагнення.

Інтелектуальна оркестрація робочих процесів

У сфері розробки застосунків, робочі процеси відіграють вирішальну роль у визначенні того, як структуруються та виконуються завдання, процеси та взаємодія з користувачами. Оскільки застосунки стають складнішими, а очікування користувачів продовжують зростати, потреба в інтелектуальній та адаптивній оркестрації робочих процесів стає все більш очевидною.

Підхід “Інтелектуальна оркестрація робочих процесів” зосереджується на використанні компонентів штучного інтелекту для динамічної оркестрації та оптимізації складних робочих процесів у застосунках. Мета полягає у створенні застосунків, які є більш ефективними, чутливими та адаптивними до даних реального часу та контексту.

У цьому розділі ми розглянемо ключові принципи та шаблони, що лежать в основі підходу інтелектуальної оркестрації робочих процесів. Ми розглянемо, як ШІ можна використовувати для інтелектуальної маршрутизації завдань, автоматизації прийняття рішень та динамічної адаптації робочих процесів на основі різних факторів, таких як поведінка користувачів, продуктивність системи та бізнес-правила. За допомогою практичних прикладів та реальних сценаріїв ми продемонструємо трансформаційний потенціал ШІ у спрощенні та оптимізації робочих процесів застосунків.

Незалежно від того, чи створюєте ви корпоративні застосунки зі складними бізнес-процесами чи застосунки для споживачів з динамічними користувацькими подорожами, шаблони та методи, обговорені в цьому розділі, нададуть вам знання та інструменти для створення інтелектуальних та ефективних робочих процесів, які покращують загальний користувацький досвід та створюють бізнес-цінність.

Бізнес-потреба

Традиційні підходи до управління робочими процесами часто покладаються на попередньо визначені правила та статичні дерева рішень, які можуть бути жорсткими, негнучкими та нездатними впоратися з динамічною природою сучасних застосунків.

Розглянемо сценарій, де застосунок електронної комерції повинен обробляти складний процес виконання замовлень. Робочий процес може включати кілька кроків, таких як перевірка замовлення, перевірка наявності товару, обробка платежів, доставка та сповіщення клієнтів. Кожен крок може мати власний набір правил, залежностей, зовнішніх інтеграцій та механізмів обробки винятків. Управління таким робочим процесом вручну або через жорстко закодовану логіку може швидко стати обтяжливим, схильним до помилок та складним для підтримки.

Крім того, коли застосунок масштабується і кількість одночасних користувачів зростає, робочий процес може потребувати адаптації та самооптимізації на основі даних реального часу та продуктивності системи. Наприклад, під час пікових періодів трафіку застосунок може потребувати динамічного коригування робочого процесу для пріоритезації певних завдань, ефективного розподілу ресурсів та забезпечення плавного користувацького досвіду.

Саме тут вступає в дію підхід “Інтелектуальна оркестрація робочих процесів”. Використовуючи компоненти ШІ, розробники можуть створювати робочі процеси, які є інтелектуальними, адаптивними та самооптимізуючими. ШІ може аналізувати величезні обсяги даних, вчитися на минулому досвіді та приймати обґрунтовані рішення в реальному часі для ефективної оркестрації робочого процесу.

Ключові переваги

1. **Підвищена ефективність**: ШІ може оптимізувати розподіл завдань, використання ресурсів та виконання робочих процесів, що призводить до швидшої обробки та покращення загальної ефективності.
2. **Адаптивність**: Робочі процеси, керовані ШІ, можуть динамічно адаптуватися до змінних умов, таких як коливання попиту користувачів, продуктивність системи або бізнес-вимоги, забезпечуючи чутливість та стійкість застосунку.
3. **Автоматизоване прийняття рішень**: ШІ може автоматизувати складні процеси прийняття рішень у робочому процесі, зменшуючи ручне втручання та мінімізуючи ризик людських помилок.
4. **Персоналізація**: ШІ може аналізувати поведінку користувачів, уподобання та контекст для персоналізації робочого процесу та надання індивідуального досвіду кожному користувачу.

5. **Масштабованість**: Робочі процеси, що працюють на ШІ, можуть безперешкодно масштабуватися для обробки зростаючих обсягів даних та взаємодій користувачів, не compromising продуктивність чи надійність.

У наступних розділах ми розглянемо ключові шаблони та методи, які уможливлюють впровадження інтелектуальних робочих процесів, та продемонструємо реальні приклади того, як ШІ трансформує управління робочими процесами в сучасних застосунках.

Ключові шаблони

Для впровадження інтелектуальної оркестрації робочих процесів у застосунках розробники можуть використовувати кілька ключових шаблонів, які використовують потужність ШІ. Ці шаблони забезпечують структурований підхід до проектування та управління робочими процесами, дозволяючи застосункам адаптуватися, оптимізувати та автоматизувати процеси на основі даних реального часу та контексту. Давайте розглянемо деякі фундаментальні шаблони в інтелектуальній оркестрації робочих процесів.

Динамічна маршрутизація завдань

Цей шаблон передбачає використання ШІ для інтелектуальної маршрутизації завдань у робочому процесі на основі різних факторів, таких як пріоритет завдання, доступність ресурсів та продуктивність системи. Алгоритми ШІ можуть аналізувати характеристики кожного завдання, враховувати поточний стан системи та приймати обґрунтовані рішення щодо призначення завдань найбільш відповідним ресурсам або шляхам обробки. Динамічна маршрутизація завдань забезпечує ефективний розподіл та виконання завдань, оптимізуючи загальну продуктивність робочого процесу.

```
class TaskRouter
  include Raix::ChatCompletion
  include Raix::FunctionDispatch

  attr_accessor :task

  # list of functions that can be called by the AI entirely at its
  # discretion depending on the task received

  function :analyze_task_priority do
    TaskPriorityAnalyzer.perform(task)
  end

  function :check_resource_availability, # ...
  function :assess_system_performance, # ...
  function :assign_task_to_resource, # ...

  DIRECTIVE = "You are a task router, responsible for intelligently
   assigning tasks to available resources based on priority, resource
   availability, and system performance..."

  def initialize(task)
    self.task = task
    transcript << { system: DIRECTIVE }
    transcript << { user: task.to_json }
  end

  def perform
    while task.unassigned?
      chat_completion

      # todo: add max loop counter and break
    end

    # capture the transcript for later analysis
    task.update(routing_transcript: transcript)
  end
end
```

Зверніть увагу на цикл, створений виразом while у рядку 29, який продовжує запитувати ШІ, доки завдання не буде призначено. У рядку 35 ми зберігаємо

транскрипт завдання для подальшого аналізу та зневадження, якщо це стане необхідним.

Контекстне прийняття рішень

Ви можете використовувати дуже подібний код для прийняття рішень з урахуванням контексту у робочому процесі. Аналізуючи відповідні точки даних, такі як уподобання користувачів, історичні шаблони та вхідні дані в реальному часі, компоненти ШІ можуть визначати найбільш доцільний курс дій у кожній точці прийняття рішень робочого процесу. Адаптуйте поведінку вашого робочого процесу на основі конкретного контексту кожного користувача чи сценарію, забезпечуючи персоналізований та оптимізований досвід.

Адаптивна композиція робочих процесів

Цей патерн зосереджується на динамічному складанні та коригуванні робочих процесів на основі змінних вимог або умов. ШІ може аналізувати поточний стан робочого процесу, виявляти вузькі місця або неефективності та автоматично змінювати структуру робочого процесу для оптимізації продуктивності. Адаптивна композиція робочих процесів дозволяє додаткам постійно розвиватися та вдосконалювати свої процеси без необхідності ручного втручання.

Обробка та відновлення після винятків

Обробка та відновлення після винятків є критично важливими аспектами інтелектуальної оркестрації робочих процесів. При роботі з компонентами ШІ та складними робочими процесами важливо передбачати та елегантно обробляти винятки для забезпечення стабільності та надійності системи.

Ось кілька ключових міркувань та технік для обробки та відновлення після винятків в інтелектуальних робочих процесах:

1. **Поширення винятків**: Впровадьте послідовний підхід до поширення винятків між компонентами робочого процесу. Коли виняток виникає всередині компонента, він повинен бути перехоплений, записаний у журнал та переданий до оркестратора або окремого компонента, відповідального за обробку винятків. Ідея полягає в централізації обробки винятків та запобіганні їх тихому поглинанню, а також відкритті можливостей для Інтелектуальної обробки помилок.
2. **Механізми повторних спроб**: Механізми повторних спроб допомагають підвищити стійкість робочого процесу та елегантно обробляти тимчасові збої. Обов'язково спробуйте впровадити механізми повторних спроб для тимчасових або відновлюваних винятків, таких як проблеми з мережевим підключенням або недоступність ресурсів, які можуть бути автоматично повторені після визначеної затримки. Наявність оркестратора або обробника винятків на базі ШІ означає, що ваші стратегії повторних спроб не повинні бути механічними за своєю природою, покладаючись на фіксовані алгоритми, такі як експоненційний відкат. Ви можете залишити обробку повторної спроби на "розсуд" компонента ШІ, відповідального за прийняття рішення щодо обробки винятку.
3. **Стратегії відкату**: Якщо компонент ШІ не може надати дійсну відповідь або стикається з помилкою–що є поширеним явищем, враховуючи його передовий характер–майте механізм відкату для забезпечення продовження робочого процесу. Це може включати використання значень за замовчуванням, альтернативних алгоритмів або Людини в циклі для прийняття рішень та підтримки руху робочого процесу вперед.
4. **Компенсаційні дії**: Директиви оркестратора повинні включати інструкції щодо компенсаційних дій для обробки винятків, які не можуть бути вирішені автоматично. Компенсаційні дії - це кроки, які вживаються для

скасування або пом'якшення наслідків невдалої операції. Наприклад, якщо етап обробки платежу не вдається, компенсаційною дією може бути відкат транзакції та повідомлення користувача. Компенсаційні дії допомагають підтримувати узгодженість та цілісність даних у разі винятків.

5. **Моніторинг та сповіщення про винятки**: Налаштуйте механізми моніторингу та сповіщення для виявлення та повідомлення відповідних зацікавлених сторін про критичні винятки. Оркестратор можна налаштувати на пороги та правила для запуску сповіщень, коли винятки перевищують певні межі або коли виникають певні типи винятків. Це дозволяє проактивно виявляти та вирішувати проблеми до того, як вони вплинуть на загальну систему.

Ось приклад обробки та відновлення після винятків у компоненті робочого процесу Ruby:

```
class InventoryManager
  def check_availability(order)
    begin
      # Perform inventory check logic
      inventory = Inventory.find_by(product_id: order.product_id)
      if inventory.available_quantity >= order.quantity
        return true
      else
        raise InsufficientInventoryError,
              "Insufficient inventory for product #{order.product_id}"
      end
    rescue InsufficientInventoryError => e
      # Log the exception
      logger.error("Inventory check failed: #{e.message}")

      # Retry the operation after a delay
      retry_count ||= 0
      if retry_count < MAX_RETRIES
        retry_count += 1
        sleep(RETRY_DELAY)
        retry
```

```
      else
        # Fallback to manual intervention
        NotificationService.admin("Inventory check failed: Order #{order.id}")
        return false
      end
    end
  end
end
```

У цьому прикладі компонент InventoryManager перевіряє наявність товару для певного замовлення. Якщо доступної кількості недостатньо, він викликає InsufficientInventoryError. Виняток перехоплюється, реєструється, і впроваджується механізм повторних спроб. Якщо перевищено ліміт повторних спроб, компонент переходить до ручного втручання, сповіщаючи адміністратора.

Впроваджуючи надійні механізми обробки та відновлення після винятків, ви можете забезпечити стійкість, зручність обслуговування та належну обробку неочікуваних ситуацій у ваших інтелектуальних робочих процесах.

Ці шаблони формують основу оркестрування інтелектуальних робочих процесів і можуть комбінуватися та адаптуватися відповідно до конкретних вимог різних застосунків. Використовуючи ці шаблони, розробники можуть створювати робочі процеси, які є гнучкими, стійкими та оптимізованими для продуктивності та зручності користування.

У наступному розділі ми розглянемо, як ці шаблони можна реалізувати на практиці, використовуючи приклади з реального світу та фрагменти коду для ілюстрації інтеграції компонентів ШІ в керування робочими процесами.

Практична реалізація оркестрування інтелектуальних робочих процесів

Тепер, коли ми розглянули ключові шаблони в оркеструванні інтелектуальних робочих процесів, давайте заглибимося в те, як ці шаблони можна реалізувати в реальних застосунках. Ми надамо практичні приклади та фрагменти коду для ілюстрації інтеграції компонентів ШІ в керування робочими процесами.

Інтелектуальний обробник замовлень

Давайте розглянемо практичний приклад реалізації оркестрування інтелектуальних робочих процесів, використовуючи компонент OrderProcessor на основі ШІ в застосунку електронної комерції Ruby on Rails. OrderProcessor реалізує концепцію Менеджера процесів корпоративної інтеграції, яку ми вперше зустріли в Розділі 3 при обговоренні Множини працівників. Компонент відповідатиме за керування процесом виконання замовлень, прийняття рішень щодо маршрутизації на основі проміжних результатів та оркестрування виконання різних етапів обробки.

Процес виконання замовлення включає кілька етапів, таких як перевірка замовлення, перевірка наявності, обробка платежів та доставка. Кожен етап реалізовано як окремий робочий процес, який виконує конкретне завдання і повертає результат до OrderProcessor. Етапи не є обов'язковими і навіть не обов'язково мають виконуватися в точному порядку.

Ось приклад реалізації OrderProcessor. Він включає два міксини з Raix. Перший (ChatCompletion) надає йому можливість виконувати чат-завершення, що робить його компонентом ШІ. Другий (FunctionDispatch) забезпечує виклик функцій штучним інтелектом, дозволяючи йому відповідати на запит викликом функції замість текстового повідомлення.

Робочі функції (validate_order, check_inventory тощо) делегують роботу відповідним робочим класам, які можуть бути компонентами ШІ або не-ШІ, з єдиною вимогою, щоб вони повертали результати своєї роботи у форматі, який можна представити як рядок.

Як і всі інші приклади в цій частині книги, цей код є практично псевдокодом і призначений лише для передачі сенсу шаблону та натхнення для ваших власних творінь. Повні описи шаблонів та повні приклади коду включені в Частину 2.

```
class OrderProcessor
  include Raix::ChatCompletion
  include Raix::FunctionDispatch

  SYSTEM_DIRECTIVE = "You are an order processor, tasked with..."

  def initialize(order)
    self.order = order
    transcript << { system: SYSTEM_DIRECTIVE }
    transcript << { user: order.to_json }
  end

  def perform
    # will continue looping until `stop_looping!` is called
    chat_completion(loop: true)
  end

  # list of functions available to be called by the AI
  # truncated for brevity

  def functions
    [
      {
        name: "validate_order",
        description: "Invoke to check validity of order",
        parameters: {
          ...
      },
```

```
      ...
    ]
  end

  # implementation of functions that can be called by the AI
  # entirely at its discretion, depending on the needs of the order

  def validate_order
    OrderValidationWorker.perform(@order)
  end

  def check_inventory
    InventoryCheckWorker.perform(@order)
  end

  def process_payment
    PaymentProcessingWorker.perform(@order)
  end

  def schedule_shipping
    ShippingSchedulerWorker.perform(@order)
  end

  def send_confirmation
    OrderConfirmationWorker.perform(@order)
  end

  def finished_processing
    @order.update!(transcript:, processed_at: Time.current)
    stop_looping!
  end
end
```

У прикладі OrderProcessor ініціалізується з об'єктом замовлення та веде протокол виконання робочого процесу у типовому форматі діалогового транскрипту, який є властивим для великих мовних моделей. ШІ надається повний контроль над оркеструванням виконання різних етапів обробки, таких як перевірка замовлення, перевірка наявності товару, обробка платежів та доставка.

Щоразу, коли викликається метод chat_completion, транскрипт надсилається

до ШІ для отримання відповіді у вигляді виклику функції. Саме ШІ повністю відповідає за аналіз результату попереднього кроку та визначення відповідної дії. Наприклад, якщо перевірка запасів виявляє низький рівень товару, OrderProcessor може запланувати завдання поповнення. Якщо обробка платежу не вдається, він може ініціювати повторну спробу або повідомити службу підтримки клієнтів.

> У наведеному вище прикладі не визначено функції для поповнення запасів чи сповіщення служби підтримки клієнтів, але вони цілком могли б бути.

Транскрипт зростає щоразу, коли викликається функція, і служить записом виконання робочого процесу, включаючи результати кожного кроку та згенеровані ШІ інструкції для наступних кроків. Цей транскрипт можна використовувати для налагодження, аудиту та забезпечення прозорості процесу виконання замовлень.

Використовуючи ШІ в OrderProcessor, програма електронної комерції може динамічно адаптувати робочий процес на основі даних у реальному часі та розумно обробляти винятки. Компонент ШІ може приймати обґрунтовані рішення, оптимізувати робочий процес та забезпечувати безперебійну обробку замовлень навіть у складних сценаріях.

Той факт, що єдиною вимогою до робочих процесів є повернення зрозумілого виводу, який ШІ може врахувати при прийнятті рішення про подальші дії, може наштовхнути вас на думку про те, як цей підхід може зменшити роботу з відображення входів/виходів, яка зазвичай потрібна при інтеграції різних систем між собою.

Інтелектуальний модератор контенту

Програми соціальних мереж зазвичай потребують принаймні мінімальної модерації контенту для забезпечення безпечної та здорової спільноти. Цей приклад компонента ContentModerator використовує ШІ для інтелектуального оркестрування процесу модерації, приймаючи рішення на основі характеристик контенту та результатів різних етапів модерації.

Процес модерації включає кілька етапів, таких як аналіз тексту, розпізнавання зображень, оцінка репутації користувача та ручний перегляд. Кожен етап реалізований як окремий робочий процес, який виконує конкретне завдання та повертає результат до ContentModerator.

Ось приклад реалізації ContentModerator:

```
class ContentModerator
  include Raix::ChatCompletion
  include Raix::FunctionDispatch

  SYSTEM_DIRECTIVE = "You are a content moderator process manager,
    tasked with the workflow involved in moderating user-generated content..."

  def initialize(content)
    @content = content
    @transcript = [
      { system: SYSTEM_DIRECTIVE },
      { user: content.to_json }
    ]
  end

  def perform
    complete(@transcript)
  end

  def model
    "openai/gpt-4"
  end

```

```
  # list of functions available to be called by the AI
  # truncated for brevity

  def functions
    [
      {
        name: "analyze_text",
        # ...
      },
      {
        name: "recognize_image",
        description: "Invoke to describe images...",
        # ...
      },
      {
        name: "assess_user_reputation",
        # ...
      },
      {
        name: "escalate_to_manual_review",
        # ...
      },
      {
        name: "approve_content",
        # ...
      },
      {
        name: "reject_content",
        # ...
      }
    ]
  end

  # implementation of functions that can be called by the AI
  # entirely at its discretion, depending on the needs of the order

  def analyze_text
    result = TextAnalysisWorker.perform(@content)
    continue_with(result)
  end

  def recognize_image
```

```
    result = ImageRecognitionWorker.perform(@content)
    continue_with(result)
  end

  def assess_user_reputation
    result = UserReputationWorker.perform(@content.user)
    continue_with(result)
  end

  def escalate_to_manual_review
    ManualReviewWorker.perform(@content)
    @content.update!(status: 'pending', transcript: @transcript)
  end

  def approve_content
    @content.update!(status: 'approved', transcript: @transcript)
  end

  def reject_content
    @content.update!(status: 'rejected', transcript: @transcript)
  end

  private

  def continue_with(result)
    @transcript << { function: result }
    complete(@transcript)
  end
end
```

У цьому прикладі ContentModerator ініціалізується з об'єктом вмісту та веде протокол модерації у форматі розмови. Компонент ШІ має повний контроль над процесом модерації, вирішуючи, які кроки виконувати на основі характеристик вмісту та результатів кожного кроку.

Доступні робочі функції, які ШІ може викликати, включають analyze_text, recognize_image, assess_user_reputation та escalate_to_manual_review. Кожна функція делегує завдання відповідному робочому процесу (TextAnalysisWorker, ImageRecognitionWorker тощо) і додає результат до протоколу модерації,

за винятком функції ескалації, яка діє як кінцевий стан. Нарешті, функції approve_content та reject_content також діють як кінцеві стани.

Компонент ШІ аналізує вміст і визначає відповідні дії. Якщо вміст містить посилання на зображення, він може викликати робочий процес recognize_image для допомоги з візуальним оглядом. Якщо будь-який робочий процес попереджає про потенційно шкідливий вміст, ШІ може вирішити передати вміст на ручний огляд або просто відхилити його. Але залежно від серйозності попередження, ШІ може вирішити використовувати результати оцінки репутації користувача при прийнятті рішення щодо вмісту, в якому він не впевнений. Залежно від випадку використання, можливо, довіреним користувачам надається більше свободи в тому, що вони можуть публікувати. І так далі, і тому подібне…

Як і в попередньому прикладі менеджера процесів, протокол модерації служить записом виконання робочого процесу, включаючи результати кожного кроку та рішення, згенеровані ШІ. Цей протокол можна використовувати для аудиту, прозорості та вдосконалення процесу модерації з часом.

Використовуючи ШІ в ContentModerator, соціальна медіа-програма може динамічно адаптувати процес модерації на основі характеристик вмісту та розумно обробляти складні сценарії модерації. Компонент ШІ може приймати обґрунтовані рішення, оптимізувати робочий процес та забезпечувати безпечний та здоровий досвід спільноти.

Розглянемо ще два приклади, які демонструють прогнозоване планування завдань та обробку винятків і відновлення в контексті інтелектуальної оркестрації робочих процесів.

Прогнозоване планування завдань у системі підтримки клієнтів

У програмі підтримки клієнтів, створеній за допомогою Ruby on Rails, ефективне управління та пріоритизація заявок підтримки є критично важливими для

надання своєчасної допомоги клієнтам. Компонент SupportTicketScheduler використовує ШІ для прогнозованого планування та призначення заявок підтримки доступним агентам на основі різних факторів, таких як терміновість заявки, експертиза агента та робоче навантаження.

```
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
```

```
79
80    def continue_with(result)
81      @transcript << { function: result }
82      complete(@transcript)
83    end
84  end
```

У цьому прикладі, SupportTicketScheduler ініціалізується з об'єктом заявки технічної підтримки та веде журнал планування. ШІ-компонент аналізує деталі заявки та прогностично планує призначення заявки на основі таких факторів, як терміновість заявки, компетенція агента та прогнозоване навантаження на агента.

Доступні функції, які може викликати ШІ, включають analyze_ticket_urgency, list_available_agents, predict_agent_workload та assign_ticket_to_agent. Кожна функція делегує завдання відповідному компоненту аналізатора чи предиктора та додає результат до журналу планування. ШІ також має можливість відкласти призначення за допомогою функції delay_assignment.

ШІ-компонент вивчає журнал планування та приймає обґрунтовані рішення щодо призначення заявок. Він враховує терміновість заявки, компетенцію доступних агентів та прогнозоване навантаження на кожного агента, щоб визначити найбільш підходящого агента для обробки заявки.

Використовуючи прогностичне планування завдань, система підтримки клієнтів може оптимізувати призначення заявок, зменшити час відгуку та покращити загальне задоволення клієнтів. Проактивне та ефективне управління заявками технічної підтримки забезпечує призначення правильних заявок правильним агентам у правильний час.

Обробка винятків та відновлення в конвеєрі обробки даних

Обробка винятків та відновлення після збоїв є важливими для забезпечення цілісності даних та запобігання їх втрати. Компонент DataProcessingOrchestrator

використовує ШІ для інтелектуальної обробки винятків та оркестрації процесу відновлення в конвеєрі обробки даних

```
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
```

У цьому прикладі DataProcessingOrchestrator ініціалізується з об'єктом пакету даних та веде протокол обробки. Компонент ШІ керує конвеєром обробки даних, опрацьовуючи винятки та відновлюючись після збоїв за потреби.

Доступні для виклику ШІ функції включають validate_data, process_data, request_fix, retry_processing та mark_data_as_failed. Кожна функція делегує завдання відповідному компоненту обробки даних і додає результат або деталі винятку до протоколу обробки.

Якщо під час кроку validate_data виникає виняток валідації, функція handle_-validation_exception додає дані про виняток до протоколу і повертає керування до ШІ. Аналогічно, якщо виняток обробки виникає під час кроку process_data, ШІ може вирішити, яку стратегію відновлення обрати.

Залежно від характеру виявленого винятку, ШІ може на свій розсуд вирішити

викликати request_fix, який делегує завдання компоненту SmartDataFixer на основі ШІ (див. розділ про Самовідновлювані дані). Засіб виправлення даних отримує простий текстовий опис того, як він повинен модифікувати @data_batch, щоб можна було повторити обробку. Можливо, успішне повторення передбачатиме видалення записів з пакету даних, які не пройшли перевірку, та/або копіювання їх до іншого конвеєра обробки для перегляду людиною? Можливості тут практично безмежні.

Завдяки впровадженню обробки винятків та відновлення на основі ШІ, програма обробки даних стає більш стійкою та відмовостійкою. DataProcessingOrchestrator розумно керує винятками, мінімізує втрату даних та забезпечує плавне виконання робочого процесу обробки даних.

Моніторинг та журналювання

Моніторинг та журналювання забезпечують видимість прогресу, продуктивності та стану компонентів робочого процесу на основі ШІ, дозволяючи розробникам відстежувати та аналізувати поведінку системи. Впровадження ефективних механізмів моніторингу та журналювання є важливим для налагодження, аудиту та постійного вдосконалення інтелектуальних робочих процесів.

Моніторинг прогресу та продуктивності робочого процесу

Для забезпечення плавного виконання інтелектуальних робочих процесів важливо відстежувати прогрес та продуктивність кожного компонента робочого процесу. Це включає відстеження ключових показників та подій протягом життєвого циклу робочого процесу.

Важливі аспекти для моніторингу включають:

1. **Час виконання робочого процесу**: Вимірювання часу, який потрібен кожному компоненту робочого процесу для виконання свого завдання. Це

допомагає виявити вузькі місця продуктивності та оптимізувати загальну ефективність робочого процесу.

2. **Використання ресурсів**: Моніторинг використання системних ресурсів, таких як CPU, пам'ять та сховище, кожним компонентом робочого процесу. Це допомагає забезпечити роботу системи в межах її можливостей та ефективну обробку навантаження.

3. **Частота помилок та винятків**: Відстеження виникнення помилок та винятків у компонентах робочого процесу. Це допомагає виявити потенційні проблеми та забезпечує проактивну обробку помилок та відновлення.

4. **Точки прийняття рішень та результати**: Моніторинг точок прийняття рішень у робочому процесі та результатів рішень, прийнятих ШІ. Це надає розуміння поведінки та ефективності компонентів ШІ.

Дані, зібрані процесами моніторингу, можуть відображатися на інформаційних панелях або використовуватися як вхідні дані для запланованих звітів, які інформують системних адміністраторів про стан системи.

Дані моніторингу можуть подаватися до процесу системного адміністратора на основі ШІ для перегляду та потенційних дій!

Журналювання ключових подій та рішень

Журналювання є важливою практикою, яка включає захоплення та зберігання відповідної інформації про ключові події, рішення та винятки, що виникають під час виконання робочого процесу.

Важливі аспекти для журналювання включають:

1. **Ініціація та завершення робочого процесу**: Журналювання часу початку та закінчення кожного екземпляру робочого процесу, разом з відповідними метаданими, такими як вхідні дані та контекст користувача.

2. **Виконання компонентів**: Журналювання деталей виконання кожного компонента робочого процесу, включаючи вхідні параметри, вихідні результати та будь-які проміжні дані, що генеруються.

3. **Рішення ШІ та обґрунтування**: Журналювання рішень, прийнятих компонентами ШІ, разом з основним обґрунтуванням або показниками впевненості. Це забезпечує прозорість та можливість аудиту рішень, прийнятих ШІ.

4. **Винятки та повідомлення про помилки**: Журналювання будь-яких винятків або повідомлень про помилки, виявлених під час виконання робочого процесу, включаючи трасування стеку та відповідну контекстну інформацію.

Журналювання може бути реалізовано за допомогою різних технік, таких як запис у файли журналу, зберігання журналів у базі даних або надсилання журналів до централізованої служби журналювання. Важливо вибрати систему журналювання, яка забезпечує гнучкість, масштабованість та легку інтеграцію з архітектурою програми.

Ось приклад того, як можна реалізувати журналювання в програмі Ruby on Rails за допомогою класу ActiveSupport::Logger:

```
class WorkflowLogger
  def self.log(message, severity = :info)
    @logger ||= ActiveSupport::Logger.new('workflow.log')
    @logger.formatter ||= proc do |severity, datetime, progname, msg|
      "#{datetime} [#{severity}] #{msg}\n"
    end
    @logger.send(severity, message)
  end
end

# Usage example
WorkflowLogger.log("Workflow initiated for order ##{@order.id}")
WorkflowLogger.log("Payment processing completed successfully")
WorkflowLogger.log("Inventory check failed for item ##{item.id}", :error)
```

Стратегічне розміщення операторів журналювання в компонентах робочого процесу та точках прийняття рішень ШІ дозволяє розробникам збирати важливу інформацію для зневадження, аудиту та аналізу.

Переваги моніторингу та журналювання

Впровадження моніторингу та журналювання в інтелектуальній оркестрації робочих процесів має кілька переваг:

1. **Зневадження та усунення несправностей**: Детальні журнали та дані моніторингу допомагають розробникам швидко виявляти та діагностувати проблеми. Вони надають розуміння потоку виконання робочого процесу, взаємодії компонентів та будь-яких помилок чи винятків, що виникають.

2. **Оптимізація продуктивності**: Моніторинг показників продуктивності дозволяє розробникам виявляти вузькі місця та оптимізувати компоненти робочого процесу для кращої ефективності. Аналізуючи час виконання, використання ресурсів та інші метрики, розробники можуть приймати обґрунтовані рішення для покращення загальної продуктивності системи.

3. **Аудит та відповідність вимогам**: Журналювання ключових подій та рішень забезпечує журнал аудиту для регуляторної відповідності та підзвітності. Це дозволяє організаціям відстежувати та перевіряти дії, виконані компонентами ШІ, та забезпечувати дотримання бізнес-правил і законодавчих вимог.

4. **Постійне вдосконалення**: Дані моніторингу та журналювання служать цінними вхідними даними для постійного вдосконалення інтелектуальних робочих процесів. Аналізуючи історичні дані, виявляючи закономірності та вимірюючи ефективність рішень ШІ, розробники можуть ітеративно вдосконалювати та покращувати логіку оркестрації робочих процесів.

Міркування та найкращі практики

При впровадженні моніторингу та журналювання в інтелектуальній оркестрації робочих процесів варто врахувати такі найкращі практики:

1. **Визначення чітких метрик моніторингу**: Визначте ключові метрики та події, які потрібно відстежувати, виходячи з конкретних вимог робочого процесу. Зосередьтеся на метриках, які надають значущу інформацію про продуктивність, стан та поведінку системи.

2. **Впровадження детального журналювання**: Переконайтеся, що оператори журналювання розміщені у відповідних точках компонентів робочого процесу та точках прийняття рішень ШІ. Фіксуйте відповідну контекстну інформацію, таку як вхідні параметри, результати виведення та будь-які проміжні дані.

3. **Використання структурованого журналювання**: Застосовуйте формат структурованого журналювання для полегшення аналізу та обробки даних журналу. Структуроване журналювання забезпечує кращий пошук, фільтрацію та агрегацію записів журналу.

4. **Керування збереженням та ротацією журналів**: Впровадьте політики збереження та ротації для керування зберіганням та життєвим циклом файлів журналів. Визначте відповідний період зберігання на основі законодавчих вимог, обмежень зберігання та потреб аналізу. Якщо можливо, передайте журналювання сторонньому сервісу, такому як Papertrail.

5. **Захист конфіденційної інформації**: Будьте обережні при журналюванні конфіденційної інформації, такої як персональні дані (PII) або конфіденційні бізнес-дані. Впровадьте відповідні заходи безпеки, такі як маскування даних або шифрування, для захисту конфіденційної інформації у файлах журналів.

6. **Інтеграція з інструментами моніторингу та сповіщення**: Використовуйте інструменти моніторингу та сповіщення для централізованого збору, аналізу та візуалізації даних моніторингу та журналювання. Ці інструменти можуть

надавати інформацію в реальному часі, генерувати сповіщення на основі попередньо визначених порогових значень та сприяти проактивному виявленню та вирішенню проблем. Мій улюблений з цих інструментів - Datadog.

Впроваджуючи комплексні механізми моніторингу та журналювання, розробники можуть отримати цінну інформацію про поведінку та продуктивність інтелектуальних робочих процесів. Ця інформація дозволяє ефективно зневаджувати, оптимізувати та постійно вдосконалювати системи оркестрації робочих процесів на основі ШІ.

Міркування щодо масштабованості та продуктивності

Масштабованість та продуктивність є критичними аспектами, які слід враховувати при проектуванні та впровадженні систем інтелектуальної оркестрації робочих процесів. Оскільки обсяг паралельних робочих процесів та складність компонентів на основі ШІ зростають, стає важливим забезпечити ефективну обробку навантаження системою та її безперебійне масштабування відповідно до зростаючих потреб.

Обробка великих обсягів паралельних робочих процесів

Системи інтелектуальної оркестрації робочих процесів часто повинні обробляти велику кількість паралельних процесів. Для забезпечення масштабованості розгляньте такі стратегії:

1. **Асинхронна обробка**: Впровадьте механізми асинхронної обробки для розділення виконання компонентів робочого процесу. Це дозволяє системі обробляти кілька робочих процесів одночасно без блокування або очікування завершення кожного компонента. Асинхронна обробка може бути реалізована

за допомогою черг повідомлень, архітектур на основі подій або фреймворків обробки фонових завдань, таких як Sidekiq.

2. **Розподілена архітектура**: Спроектуйте архітектуру системи для використання безсерверних компонентів (таких як AWS Lambda) або просто розподіліть навантаження між кількома вузлами чи серверами поряд з вашим основним сервером додатків. Це забезпечує горизонтальну масштабованість, де можна додавати додаткові вузли для обробки збільшених обсягів робочих процесів.

3. **Паралельне виконання**: Визначте можливості для паралельного виконання в робочих процесах. Деякі компоненти робочого процесу можуть бути незалежними один від одного і можуть виконуватися одночасно. Використовуючи методи паралельної обробки, такі як багатопотоковість або розподілені черги завдань, система може оптимізувати використання ресурсів та зменшити загальний час виконання робочого процесу.

Оптимізація продуктивності компонентів на базі ШІ

Компоненти на базі штучного інтелекту, такі як моделі машинного навчання чи системи обробки природної мови, можуть бути обчислювально інтенсивними та впливати на загальну продуктивність системи оркестрації робочих процесів. Щоб оптимізувати продуктивність компонентів ШІ, розгляньте такі методи:

1. **Кешування**: Якщо ваша обробка ШІ є суто генеративною і не включає пошук інформації в реальному часі або зовнішні інтеграції для генерації відповідей чату, тоді ви можете розглянути механізми кешування для зберігання та повторного використання результатів часто запитуваних або обчислювально затратних операцій.

2. **Оптимізація моделі**: Постійно оптимізуйте спосіб використання моделей ШІ в компонентах робочого процесу. Це може включати такі методи, як

Дистиляція промптів, або може просто полягати в тестуванні нових моделей, коли вони стають доступними.

3. **Пакетна обробка**: Якщо ви працюєте з моделями класу GPT-4, ви можете використовувати методи пакетної обробки для обробки кількох точок даних або запитів в одному пакеті, замість їх індивідуальної обробки. Обробляючи дані пакетами, система може оптимізувати використання ресурсів та зменшити накладні витрати повторних запитів до моделі.

Моніторинг та профілювання продуктивності

Для виявлення вузьких місць продуктивності та оптимізації масштабованості системи інтелектуальної оркестрації робочих процесів, важливо впровадити механізми моніторингу та профілювання. Розгляньте такі підходи:

1. **Показники продуктивності**: Визначте та відстежуйте ключові показники продуктивності, такі як час відгуку, пропускна здатність, використання ресурсів та затримка. Ці показники надають розуміння продуктивності системи та допомагають визначити області для оптимізації. Популярний агрегатор моделей ШІ OpenRouter включає метрики Host[1] та Speed[2] у кожній відповіді API, що робить відстеження цих ключових показників тривіальним.

2. **Інструменти профілювання**: Використовуйте інструменти профілювання для аналізу продуктивності окремих компонентів робочого процесу та операцій ШІ. Інструменти профілювання можуть допомогти визначити гарячі точки продуктивності, неефективні шляхи коду або ресурсомісткі операції. Популярні інструменти профілювання включають New Relic, Scout або вбудовані профілювальники, що надаються мовою програмування чи фреймворком.

[1]Host - це час, який знадобився для отримання першого байту потокової генерації від хоста моделі, тобто "час до першого байту."

[2]Speed розраховується як кількість токенів завершення, поділена на загальний час генерації. Для непотокових запитів затримка вважається частиною часу генерації.

3. **Навантажувальне тестування**: Проводьте навантажувальне тестування для оцінки продуктивності системи при різних рівнях паралельного навантаження. Навантажувальне тестування допомагає визначити межі масштабованості системи, виявити погіршення продуктивності та переконатися, що система може обробляти очікуваний трафік без погіршення продуктивності.

4. **Постійний моніторинг**: Впровадьте механізми постійного моніторингу та сповіщення для проактивного виявлення проблем з продуктивністю та вузьких місць. Налаштуйте панелі моніторингу та сповіщення для відстеження ключових показників ефективності (KPI) та отримання повідомлень при порушенні попередньо визначених порогів. Це дозволяє швидко виявляти та вирішувати проблеми з продуктивністю.

Стратегії масштабування

Щоб впоратися зі зростаючими навантаженнями та забезпечити масштабованість системи інтелектуальної оркестрації робочих процесів, розгляньте такі стратегії масштабування:

1. **Вертикальне масштабування**: Вертикальне масштабування передбачає збільшення ресурсів (наприклад, CPU, пам'яті) окремих вузлів або серверів для обробки більших навантажень. Цей підхід підходить, коли система потребує більше обчислювальної потужності або пам'яті для обробки складних робочих процесів або операцій ШІ.

2. **Горизонтальне масштабування**: Горизонтальне масштабування передбачає додавання більшої кількості вузлів або серверів до системи для розподілу навантаження. Цей підхід ефективний, коли система повинна обробляти велику кількість паралельних робочих процесів або коли навантаження може бути легко розподілене між кількома вузлами. Горизонтальне масштабування вимагає розподіленої архітектури та механізмів балансування навантаження для забезпечення рівномірного розподілу трафіку.

3. **Автоматичне масштабування**: Впровадьте механізми автоматичного масштабування для автоматичного регулювання кількості вузлів або ресурсів залежно від потреб навантаження. Автоматичне масштабування дозволяє системі динамічно масштабуватися вгору або вниз залежно від вхідного трафіку, забезпечуючи оптимальне використання ресурсів та економічну ефективність. Хмарні платформи, такі як Amazon Web Services (AWS) або Google Cloud Platform (GCP), надають можливості автоматичного масштабування, які можна використовувати для систем інтелектуальної оркестрації робочих процесів.

Методи оптимізації продуктивності

На додаток до стратегій масштабування, розгляньте такі методи оптимізації продуктивності для підвищення ефективності системи інтелектуальної оркестрації робочих процесів:

1. **Ефективне зберігання та отримання даних**: Оптимізуйте механізми зберігання та отримання даних, що використовуються компонентами робочого процесу. Використовуйте ефективне індексування бази даних, методи оптимізації запитів та кешування даних для мінімізації затримки та покращення продуктивності операцій з інтенсивним використанням даних.

2. **Асинхронний ввід/вивід**: Використовуйте асинхронні операції вводу/виводу для запобігання блокування та покращення відгуку системи. Асинхронний ввід/вивід дозволяє системі обробляти декілька запитів одночасно без очікування завершення операцій вводу/виводу, тим самим максимізуючи використання ресурсів.

3. **Ефективна серіалізація та десеріалізація**: Оптимізуйте процеси серіалізації та десеріалізації, що використовуються для обміну даними між компонентами робочого процесу. Використовуйте ефективні формати серіалізації, такі як Protocol Buffers або MessagePack, щоб зменшити накладні витрати на серіалізацію даних та покращити продуктивність міжкомпонентної комунікації.

Для програм на основі Ruby розгляньте можливість використання Universal ID. Universal ID використовує як MessagePack, так і Brotli (комбінація, створена для швидкості та найкращого у своєму класі стиснення даних). У поєднанні ці бібліотеки працюють до 30% швидше і забезпечують стиснення в межах 2-5% порівняно з Protocol Buffers.

4. **Стиснення та кодування**: Застосовуйте методи стиснення та кодування для зменшення розміру даних, що передаються між компонентами робочого процесу. Алгоритми стиснення, такі як gzip або Brotli, можуть значно зменшити використання пропускної здатності мережі та покращити загальну продуктивність системи.

Враховуючи аспекти масштабованості та продуктивності під час проєктування та впровадження систем оркестрації інтелектуальних робочих процесів, ви можете забезпечити здатність вашої системи обробляти великі обсяги паралельних робочих процесів, оптимізувати продуктивність компонентів на основі ШІ та безперешкодно масштабуватися відповідно до зростаючих потреб. Постійний моніторинг, профілювання та оптимізація є важливими для підтримки продуктивності та відгуку системи при збільшенні навантаження та складності з часом.

Тестування та валідація робочих процесів

Тестування та валідація є критично важливими аспектами розробки та підтримки систем оркестрації інтелектуальних робочих процесів. Враховуючи складну природу робочих процесів на основі ШІ, важливо забезпечити, щоб кожен компонент функціонував згідно з очікуваннями, загальний робочий процес поводився правильно, а рішення ШІ були точними та надійними. У цьому розділі ми розглянемо різні методи та міркування щодо тестування та валідації інтелектуальних робочих процесів.

Модульне тестування компонентів робочого процесу

Модульне тестування включає тестування окремих компонентів робочого процесу ізольовано для перевірки їх коректності та надійності. При модульному тестуванні компонентів на основі ШІ враховуйте наступне:

1. **Валідація вхідних даних**: Тестуйте здатність компонента обробляти різні типи вхідних даних, включаючи коректні та некоректні дані. Переконайтеся, що компонент коректно обробляє граничні випадки та надає відповідні повідомлення про помилки або винятки.

2. **Перевірка вихідних даних**: Підтверджуйте, що компонент видає очікуваний результат для заданого набору вхідних даних. Порівнюйте фактичний результат з очікуваним для забезпечення коректності.

3. **Обробка помилок**: Тестуйте механізми обробки помилок компонента, симулюючи різні сценарії помилок, такі як некоректні вхідні дані, недоступність ресурсів або неочікувані винятки. Переконайтеся, що компонент належним чином перехоплює та обробляє помилки.

4. **Граничні умови**: Тестуйте поведінку компонента в граничних умовах, таких як порожні вхідні дані, максимальний розмір вхідних даних або екстремальні значення. Переконайтеся, що компонент коректно обробляє ці умови без збоїв або видачі неправильних результатів.

Ось приклад модульного тесту для компонента робочого процесу на Ruby з використанням фреймворку тестування RSpec:

```
RSpec.describe OrderValidator do
  describe '#validate' do
    context 'when order is valid' do
      let(:order) { build(:order) }

      it 'returns true' do
        expect(subject.validate(order)).to be true
      end
    end

    context 'when order is invalid' do
      let(:order) { build(:order, total_amount: -100) }

      it 'returns false' do
        expect(subject.validate(order)).to be false
      end
    end
  end
end
```

У цьому прикладі компонент OrderValidator тестується за допомогою двох тестових випадків: одного для дійсного замовлення та іншого для недійсного замовлення. Тестові випадки перевіряють, що метод validate повертає очікуване логічне значення на основі валідності замовлення.

Інтеграційне тестування взаємодій робочого процесу

Інтеграційне тестування зосереджується на перевірці взаємодій та потоку даних між різними компонентами робочого процесу. Воно забезпечує безперебійну спільну роботу компонентів та отримання очікуваних результатів. При інтеграційному тестуванні інтелектуальних робочих процесів варто враховувати наступне:

1. Взаємодія компонентів: Тестування комунікації та обміну даними між компонентами робочого процесу. Перевірка того, що вихідні дані одного компонента правильно передаються як вхідні дані до наступного компонента в робочому процесі.

2. Узгодженість даних: Забезпечення того, що дані залишаються узгодженими та точними під час проходження через робочий процес. Перевірка правильності виконання перетворень даних, обчислень та агрегацій.

3. Поширення винятків: Тестування того, як винятки та помилки поширюються та обробляються між компонентами робочого процесу. Перевірка того, що винятки перехоплюються, реєструються та належним чином обробляються для запобігання порушення робочого процесу.

4**. Асинхронна поведінка:** Якщо робочий процес включає асинхронні компоненти або паралельне виконання, тестування механізмів координації та синхронізації. Забезпечення правильної поведінки робочого процесу в умовах паралельних та асинхронних сценаріїв.

Ось приклад інтеграційного тесту для робочого процесу в Ruby з використанням фреймворку тестування RSpec:

```
RSpec.describe OrderProcessingWorkflow do

  let(:order) { build(:order) }

  it 'processes the order successfully' do
    expect(OrderValidator).to receive(:validate).and_return(true)
    expect(InventoryManager).to receive(:check_availability).and_return(true)
    expect(PaymentProcessor).to receive(:process_payment).and_return(true)
    expect(ShippingService).to receive(:schedule_shipping).and_return(true)

    workflow = OrderProcessingWorkflow.new(order)
    result = workflow.process

    expect(result).to be true
    expect(order.status).to eq('processed')
  end

end
```

У цьому прикладі OrderProcessingWorkflow тестується шляхом перевірки взаємодій між різними компонентами робочого процесу. Тестовий випадок

встановлює очікування щодо поведінки кожного компонента та забезпечує успішну обробку замовлення робочим процесом, відповідно оновлюючи статус замовлення.

Тестування точок прийняття рішень ШІ

Тестування точок прийняття рішень ШІ є критично важливим для забезпечення точності та надійності робочих процесів на основі штучного інтелекту. При тестуванні точок прийняття рішень ШІ враховуйте наступне:

1. **Точність рішень**: Перевірте, що компонент ШІ приймає точні рішення на основі вхідних даних та навченої моделі. Порівняйте рішення ШІ з очікуваними результатами або еталонними даними.

2. **Граничні випадки**: Протестуйте поведінку компонента ШІ в граничних випадках та незвичайних сценаріях. Перевірте, що компонент ШІ коректно обробляє ці випадки та приймає обґрунтовані рішення.

3. **Упередженість та справедливість**: Оцініть компонент ШІ на наявність потенційної упередженості та переконайтеся, що він приймає справедливі та неупереджені рішення. Протестуйте компонент з різноманітними вхідними даними та проаналізуйте результати на наявність дискримінаційних шаблонів.

4. **Пояснюваність**: Якщо компонент ШІ надає пояснення або обґрунтування своїх рішень, перевірте правильність та чіткість пояснень. Переконайтеся, що пояснення узгоджуються з базовим процесом прийняття рішень.

Ось приклад тестування точки прийняття рішень ШІ в Ruby з використанням фреймворка тестування RSpec:

```
RSpec.describe FraudDetector do
  describe '#detect_fraud' do
    context 'when transaction is fraudulent' do
      let(:tx) do
        build(:transaction, amount: 10_000, location: 'High-Risk Country')
      end

      it 'returns true' do
        expect(subject.detect_fraud(tx)).to be true
      end
    end

    context 'when transaction is legitimate' do
      let(:tx) do
        build(:transaction, amount: 100, location: 'Low-Risk Country')
      end

      it 'returns false' do
        expect(subject.detect_fraud(tx)).to be false
      end
    end
  end
end
```

У цьому прикладі компонент ШІ FraudDetector тестується за допомогою двох тестових випадків: один для шахрайської транзакції, а інший для легітимної транзакції. Тестові випадки перевіряють, що метод detect_fraud повертає очікуване логічне значення на основі характеристик транзакції.

Наскрізне тестування

Наскрізне тестування включає тестування всього робочого процесу від початку до кінця, симулюючи реальні сценарії та взаємодію з користувачем. Воно забезпечує правильну поведінку робочого процесу та отримання бажаних результатів. При виконанні наскрізного тестування для інтелектуальних робочих процесів враховуйте наступне:

1. **Сценарії користувача**: Визначте поширені сценарії користувача та протестуйте поведінку робочого процесу в цих сценаріях. Перевірте, що робочий процес правильно обробляє користувацькі введення, приймає відповідні рішення та видає очікувані результати.

2. **Перевірка даних**: Переконайтеся, що робочий процес перевіряє та очищує користувацькі введення для запобігання невідповідності даних або вразливостей безпеки. Протестуйте робочий процес з різними типами вхідних даних, включаючи коректні та некоректні дані.

3. **Відновлення після помилок**: Протестуйте здатність робочого процесу відновлюватися після помилок та виключень. Симулюйте сценарії помилок та перевірте, що робочий процес коректно їх обробляє, записує помилки та виконує відповідні дії для відновлення.

4. **Продуктивність та масштабованість**: Оцініть продуктивність та масштабованість робочого процесу за різних умов навантаження. Протестуйте робочий процес з великою кількістю одночасних запитів та виміряйте час відгуку, використання ресурсів та загальну стабільність системи.

Ось приклад наскрізного тесту для робочого процесу на Ruby з використанням тестового фреймворку RSpec та бібліотеки Capybara для симуляції взаємодії з користувачем:

```
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
```

У цьому прикладі наскрізний тест імітує користувача, який розміщує замовлення через веб-інтерфейс. Він заповнює необхідні поля форми, надсилає замовлення та перевіряє, що замовлення успішно обробляється, відображаючи відповідне повідомлення про підтвердження та оновлюючи статус замовлення в базі даних.

Безперервна інтеграція та розгортання

Для забезпечення надійності та підтримки інтелектуальних робочих процесів рекомендується інтегрувати тестування та валідацію в конвеєр безперервної інтеграції та розгортання (CI/CD). Це дозволяє автоматизувати тестування та валідацію змін робочого процесу перед їх розгортанням у продакшені. Розгляньте такі практики:

1. **Автоматизоване виконання тестів**: Налаштуйте конвеєр CI/CD для автоматичного запуску набору тестів щоразу, коли вносяться зміни до кодової бази робочого процесу. Це забезпечує раннє виявлення будь-яких регресій або збоїв у процесі розробки.

2. **Моніторинг покриття тестами**: Вимірюйте та відстежуйте покриття тестами компонентів робочого процесу та точок прийняття рішень ШІ. Прагніть до

високого покриття тестами, щоб забезпечити ретельне тестування критичних шляхів та сценаріїв.

3. Постійний зворотний зв'язок: Інтегруйте результати тестів та метрики якості коду в процес розробки. Забезпечте постійний зворотний зв'язок для розробників щодо стану тестів, якості коду та будь-яких проблем, виявлених під час процесу CI/CD.

4. Середовища попереднього розгортання: Розгортайте робочий процес у середовищах попереднього розгортання, які максимально наближені до продакшен-середовища. Виконуйте додаткове тестування та валідацію в середовищі попереднього розгортання, щоб виявити будь-які проблеми, пов'язані з інфраструктурою, конфігурацією або інтеграцією даних.

5. Механізми відкату: Впровадьте механізми відкату на випадок збоїв розгортання або виявлення критичних проблем у продакшені. Переконайтеся, що робочий процес можна швидко повернути до попередньої стабільної версії, щоб мінімізувати простої та вплив на користувачів.

Включаючи тестування та валідацію протягом усього життєвого циклу розробки інтелектуальних робочих процесів, організації можуть забезпечити надійність, точність та підтримуваність своїх систем на базі ШІ. Регулярне тестування та валідація допомагають виявляти помилки, запобігати регресіям та підвищувати впевненість у поведінці та результатах робочого процесу.

Частина 2: Шаблони

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Інженерія промптів

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Ланцюжок міркувань

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Як це працює

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Приклади

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Генерація контенту

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Створення структурованих сутностей

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Інструктування агентів BMM

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Переваги та міркування

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Перемикання режиму

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Як це працює

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Коли використовувати

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Приклад

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Призначення ролі

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Як це працює

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Коли використовувати

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Приклади

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Об’єкт Запиту

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Як це працює

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Шаблон запиту

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Як це працює

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Переваги та міркування

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Коли використовувати:

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Приклад

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Структурований Введення/Виведення

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Як це працює

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Масштабування структурованого вводу-виводу

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Переваги та міркування

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Ланцюжок Промптів

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Як це працює

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Коли використовувати

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Приклад: Онбординг Olympia

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Переписувач промптів

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Як це працює

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Приклад

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Обмеження Відповідей

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Як Це Працює

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Переваги та міркування

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Обробка помилок

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Аналізатор запитів

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Як це працює

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Реалізація

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Розмітка частин мови (POS) та розпізнавання іменованих сутностей (NER)

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Класифікація намірів

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Витяг ключових слів

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Переваги

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Переписувач запитів

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Як це працює

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Приклад

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Переваги

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Ventriloquist

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Як це працює

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Коли використовувати

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Приклад

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Дискретні компоненти

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Предикат

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Як це працює

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Коли це використовувати

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Приклад

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

API-фасад

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Як це працює

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Основні переваги

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Коли використовувати

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Приклад

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Автентифікація та авторизація

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Обробка запитів

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Форматування відповідей

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Обробка помилок та крайових випадків

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Міркування щодо масштабованості та продуктивності

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Порівняння з іншими шаблонами проєктування

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Інтерпретатор результатів

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Як це працює

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Коли використовувати

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Приклад

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Віртуальна машина

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Як це працює

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Коли використовувати

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Приклад

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

За лаштунками магії

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Специфікація та тестування

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Специфікація поведінки

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Написання тестових випадків

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Приклад: Тестування компонента перекладача

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Повторення HTTP-взаємодій

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Людина в Контурі (HITL)

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Високорівневі Патерни

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Гібридний Інтелект

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Адаптивна Відповідь

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Перемикання ролей між людиною та ШІ

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Ескалація

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Як це працює

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Ключові переваги

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Практичне застосування: Охорона здоров'я

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Цикл зворотного зв’язку

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Як це працює

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Застосування та приклади

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Передові техніки інтеграції людського відгуку

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Пасивне випромінювання інформації

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Як це працює

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Контекстний показ інформації

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Проактивні сповіщення

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Пояснювальні висновки

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Інтерактивне дослідження

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Ключові переваги

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Застосування та приклади

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Спільне Прийняття Рішень (СПР)

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Як Це Працює

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Приклад

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Безперервне навчання

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Як це працює

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Застосування та приклади

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Приклад

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Етичні міркування

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Роль HITL у зменшенні ризиків ШІ

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Технологічні досягнення та майбутні перспективи

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Виклики та обмеження систем з людиною в циклі

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Інтелектуальна обробка помилок

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Традиційні підходи до обробки помилок

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Контекстна діагностика помилок

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Як це працює

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Інженерія промптів для контекстної діагностики помилок

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Генерація з розширеним пошуком для контекстної діагностики помилок

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Інтелектуальне звітування про помилки

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Превентивне запобігання помилкам

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Як це працює

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Розумне відновлення після помилок

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Як це працює

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Персоналізована комунікація про помилки

Як це працює

Адаптивний робочий процес обробки помилок

Як це працює

Контроль якості

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Eval

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Проблема

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Рішення

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Як це працює

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Приклад

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Міркування

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Розуміння золотих еталонів

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Як працюють безеталонні оцінки

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Захисний механізм

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Проблема

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Рішення

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Як це працює

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Приклад

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Міркування

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Захисні механізми та оцінювання: Дві сторони однієї медалі

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Взаємозамінність захисних механізмів та оцінювання без еталону

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Реалізація двоцільових захисних механізмів та оцінювання

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Глосарій

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Глосарій

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

A

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

B

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

C

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

D

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

E

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

F

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

G

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

H

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

I

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

J

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

K

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

L

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

M

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

N

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

O

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

P

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Q

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

R

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

S

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

T

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

U

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

V

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

W

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Z

Цей вміст недоступний у демонстраційній книзі. Книгу можна придбати на Leanpub за адресою http://leanpub.com/patterns-of-application-development-using-ai-uk.

Index