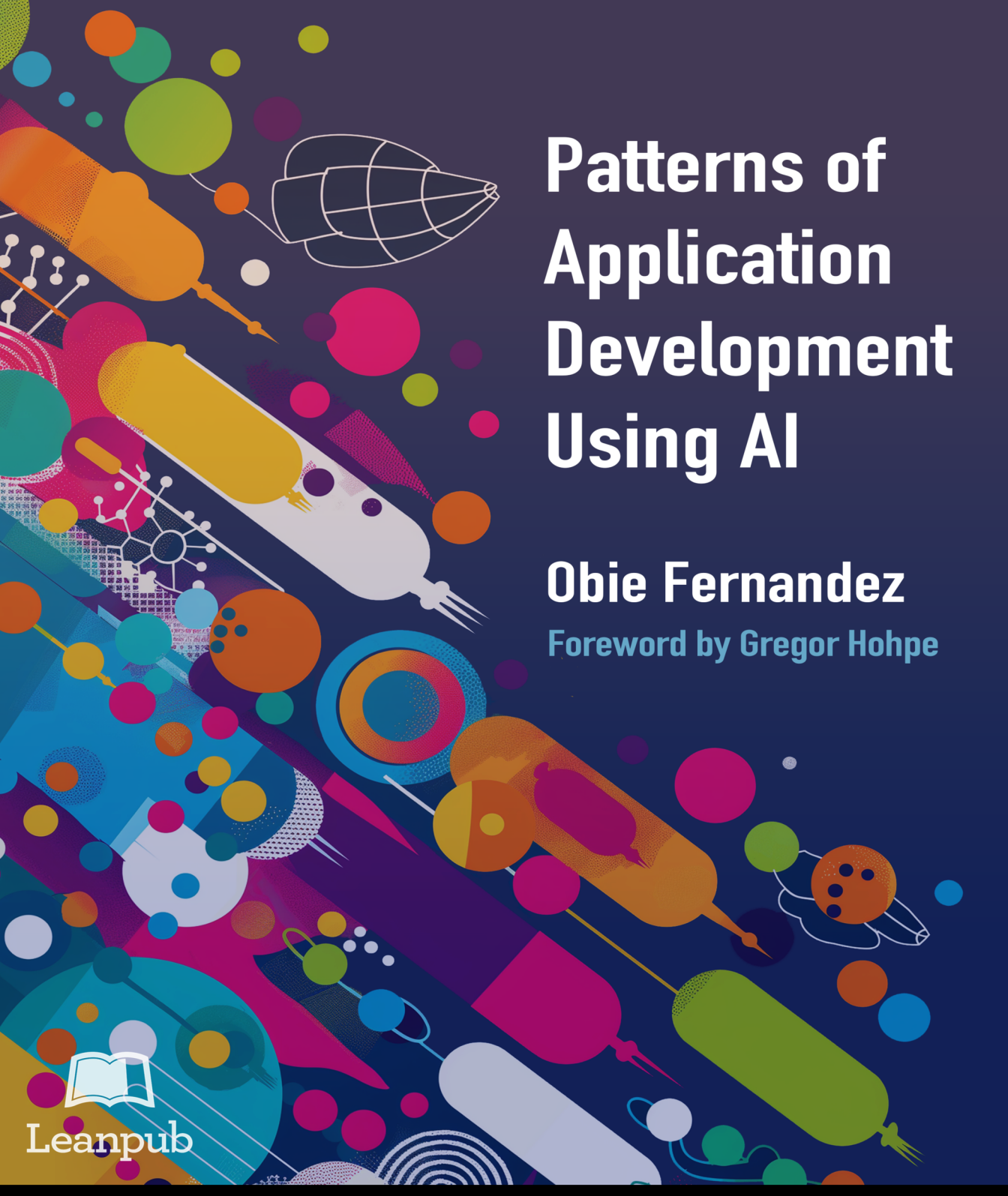


Nederlandse Editie

Patronen voor Applicatieontwikkeling met AI (Nederlandse Editie)

Obie Fernandez

Dit boek is te koop bij
http://leanpub.com/patterns-of-application-development-using-ai-nl

Deze versie is gepubliceerd op 2025-01-23

Tweet over dit boek!

Gelieve Obie Fernandez te helpen door reclame te maken over het boek Twitter!

De voorgestelde hashtag voor dit boek is #poaduai.

Lees wat andere mensen over het boek zeggen door op deze link te klikken en op Twitter naar deze hashtag te zoeken:

#poaduai

Voor mijn geweldige koningin, mijn muze, mijn licht en liefde, Victoria

Ook door 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

Inhoudsopgave

Voorwoord door Gregor Hohpe

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Voorwoord

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Over het Boek

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Over de Codevoorbeelden

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Wat Ik Niet Behandel

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Voor Wie Dit Boek Is

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Een Gemeenschappelijke Woordenschat Opbouwen

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-

ai-nl.

Betrokken Raken

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Dankwoord

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Wat is er met de illustraties?

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

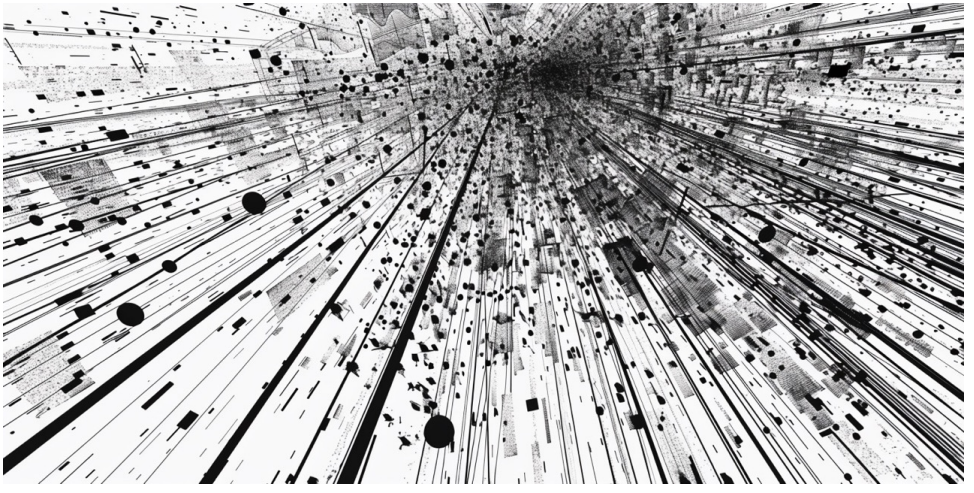
Over Lean Publishing

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Over de auteur

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Introductie



Als je staat te popelen om AI Large Language Models (LLMs) in je programmeerprojecten te integreren, kun je direct naar de patronen en codevoorbeelden in de latere hoofdstukken gaan. Om de kracht en potentie van deze patronen echter volledig te kunnen waarderen, is het de moeite waard om even stil te staan bij de bredere context en de samenhangende aanpak die ze vertegenwoordigen.

De patronen zijn niet slechts een verzameling losse technieken, maar vormen een uniform raamwerk voor het integreren van AI in je applicaties. Ik gebruik Ruby on Rails, maar deze patronen zouden in vrijwel elke andere programmeeromgeving moeten werken. Ze behandelen een breed scala aan aandachtspunten, van gegevensbeheer en prestatie-optimalisatie tot gebruikerservaring en beveiliging, en bieden daarmee een uitgebreide toolkit voor het verbeteren van traditionele programmeerpraktijken met de mogelijkheden van AI.

Elke categorie patronen pakt een specifieke uitdaging of kans aan die ontstaat bij het integreren van AI-componenten in je applicatie. Door de relaties en synergiën tussen deze patronen te begrijpen, kun je weloverwogen beslissingen nemen over waar en hoe je AI het meest effectief kunt toepassen.

Patronen zijn nooit voorschrijvende oplossingen en moeten ook niet als zodanig worden behandeld. Ze zijn bedoeld als aanpasbare bouwstenen die moeten worden afgestemd op de unieke vereisten en beperkingen van je eigen specifieke applicatie. De succesvolle toepassing van deze patronen (zoals alle andere in het softwarevakgebied) is afhankelijk van een diepgaand begrip van het probleemdomein, gebruikersbehoeften en de algehele technische architectuur van je project.

Gedachten over Softwarearchitectuur

Ik ben begonnen met programmeren in de jaren ’80 en was betrokken bij de hackerscene, en heb mijn hackersmentaliteit nooit verloren, zelfs niet nadat ik een professionele softwareontwikkelaar werd. Vanaf het begin had ik altijd een gezonde scepsis over wat softwarearchitecten in hun ivoren torens daadwerkelijk bijdroegen.

Een van de redenen waarom ik persoonlijk zo enthousiast ben over de veranderingen die deze krachtige nieuwe golf van AI-technologie teweegbrengt, is de impact ervan op wat we beschouwen als *softwarearchitectuur* beslissingen. Het daagt traditionele opvattingen uit over wat de “juiste” manier is om onze softwareprojecten te ontwerpen en implementeren. Het stelt ook ter discussie of architectuur nog steeds primair kan worden gezien als *de onderdelen van een systeem die moeilijk te veranderen zijn*, aangezien AI-verbetering het makkelijker dan ooit maakt om elk onderdeel van je project op elk moment te wijzigen.

Misschien bevinden we ons in de piekjaren van de “postmoderne” benadering van software-engineering. In deze context verwijst postmodern naar een fundamentele verschuiving weg van traditionele paradigma’s, waarbij ontwikkelaars verantwoordelijk

waren voor het schrijven en onderhouden van elke regel code. In plaats daarvan omarmt het het idee om taken, zoals gegevensmanipulatie, complexe algoritmen en zelfs hele delen van applicatielogica, te delegeren aan externe bibliotheken en API's. Deze postmoderne verschuiving vertegenwoordigt een significante afwijking van de conventionele wijsheid om applicaties vanaf de basis op te bouwen, en het daagt ontwikkelaars uit om hun rol in het ontwikkelingsproces te heroverwegen.

Ik heb altijd geloofd dat goede programmeurs alleen de code schrijven die absoluut noodzakelijk is om te schrijven, gebaseerd op de leerstellingen van Larry Wall en andere hackerluminaries zoals hij. Door de hoeveelheid geschreven code te minimaliseren, kunnen we sneller werken, het oppervlak voor bugs verkleinen, onderhoud vereenvoudigen en de algehele betrouwbaarheid van hun applicaties verbeteren. Minder code stelt ons in staat om ons te concentreren op de kernbedrijfslogica en gebruikerservaring, terwijl ander werk wordt gedelegeerd aan andere diensten.

Nu AI-aangedreven systemen taken kunnen afhandelen die voorheen exclusief het domein waren van door mensen geschreven code, zouden we nog productiever en wendbaarder moeten kunnen zijn, met meer dan ooit tevoren focus op het creëren van bedrijfswaarde en gebruikerservaring.

Natuurlijk zijn er afwegingen bij het delegeren van grote delen van je project aan AI-systemen, zoals het potentiële verlies van controle en de behoefte aan robuuste monitoring- en feedbackmechanismen. Daarom vereist het een nieuwe set vaardigheden en kennis, waaronder op zijn minst enig fundamenteel begrip van hoe AI werkt.

Wat is een Large Language Model?

Large Language Models (LLMs) zijn een type kunstmatige intelligentie model dat sinds de lancering van GPT-3 door OpenAI in 2020 aanzienlijke aandacht heeft gekregen. LLMs zijn ontworpen om menselijke taal te verwerken, begrijpen en genereren met opmerkelijke nauwkeurigheid en vloeiendheid. In deze sectie werpen we een korte blik

op hoe LLMs werken en waarom ze zo geschikt zijn voor het bouwen van intelligente systeemcomponenten.

In hun kern zijn LLMs gebaseerd op deep learning algoritmen, specifiek neurale netwerken. Deze netwerken bestaan uit onderling verbonden knooppunten, of neuronen, die informatie verwerken en doorgeven. De architectuur die vaak wordt gekozen voor LLMs is het Transformer-model, dat zeer effectief is gebleken in het verwerken van sequentiële data zoals tekst.

Transformermodellen zijn gebaseerd op het aandachtsmechanisme en worden voornamelijk gebruikt voor taken met sequentiële data, zoals natuurlijke taalverwerking. Transformers verwerken invoergegevens in één keer in plaats van sequentieel, waardoor ze langetermijnafhankelijkheden effectiever kunnen vastleggen. Ze hebben lagen van aandachtsmechanismen die het model helpen zich te concentreren op verschillende delen van de invoergegevens om context en relaties te begrijpen.

Het trainingsproces voor LLMs bestaat uit het blootstellen van het model aan enorme hoeveelheden tekstuele data, zoals boeken, artikelen, websites en code-repositories. Tijdens de training leert het model patronen, relaties en structuren binnen de tekst te herkennen. Het legt de statistische eigenschappen van de taal vast, zoals grammaticaregels, woordassociaties en contextuele betekenissen.

Een van de belangrijkste technieken die worden gebruikt bij het trainen van LLMs is ongesuperviseerd leren. Dit betekent dat het model leert van de data zonder expliciete labels of begeleiding. Het ontdekt zelfstandig patronen en representaties door het analyseren van het samen voorkomen van woorden en zinsdelen in de trainingsdata. Dit stelt LLMs in staat om een diep begrip van taal en haar complexiteit te ontwikkelen.

Een ander belangrijk aspect van LLMs is hun vermogen om *context* te verwerken. Bij het verwerken van een tekst kijken LLMs niet alleen naar de individuele woorden, maar ook naar de omringende context. Ze houden rekening met de voorgaande woorden, zinnen en zelfs paragrafen om de betekenis en intentie van de tekst te begrijpen. Dit contextuele begrip stelt LLMs in staat om samenhangende en relevante antwoorden

te genereren. Een van de belangrijkste manieren waarop we de capaciteiten van een bepaald LLM-model evalueren, is door te kijken naar de grootte van de context die ze kunnen overwegen bij het genereren van antwoorden.

Eenmaal getraind kunnen LLMs worden gebruikt voor een breed scala aan taalgerelateerde taken. Ze kunnen mensachtige tekst genereren, vragen beantwoorden, documenten samenvatten, talen vertalen en zelfs code schrijven. De veelzijdigheid van LLMs maakt ze waardevol voor het bouwen van intelligente systeemcomponenten die kunnen interacteren met gebruikers, tekstgegevens kunnen verwerken en analyseren, en betekenisvolle output kunnen genereren.

Door LLMs in de applicatiearchitectuur te integreren, kun je AI-componenten creëren die gebruikersinvoer begrijpen en verwerken, dynamische content genereren en intelligente aanbevelingen of acties kunnen doen. Maar het werken met LLMs vereist zorgvuldige overweging van resourcevereisten en prestatiecompromissen. LLMs zijn rekenintensief en kunnen aanzienlijke verwerkingskracht en geheugen (met andere woorden, geld) nodig hebben om te functioneren. De meesten van ons zullen de kostenimplicaties van het integreren van LLMs in onze applicaties moeten beoordelen en dienovereenkomstig handelen.

Inferentie Begrijpen

Inferentie verwijst naar het proces waarbij een model voorspellingen of output genereert op basis van nieuwe, niet eerder geziene data. Het is de fase waarin het getrainde model wordt gebruikt om beslissingen te nemen of tekst, afbeeldingen of andere content te genereren als reactie op gebruikersinvoer.

Tijdens de trainingsfase leert een AI-model van een grote dataset door zijn parameters aan te passen om de fout in zijn voorspellingen te minimaliseren. Eenmaal getraind kan het model wat het heeft geleerd toepassen op nieuwe data. Inferentie is hoe het model zijn geleerde patronen en kennis gebruikt om output te genereren.

Voor LLMs houdt inferentie in dat een prompt of invoertekst wordt omgezet in een coherent en contextueel relevant antwoord, als een stroom van *tokens* (waar we het binnenkort over zullen hebben). Dit kan het beantwoorden van een vraag zijn, het afmaken van een zin, het genereren van een verhaal, of het vertalen van tekst, onder vele andere taken.

In tegenstelling tot de manier waarop jij en ik denken, gebeurt het "denken" van een AI-model via inferentie allemaal in één statusloze operatie. Dat wil zeggen, zijn denken is beperkt tot zijn generatieproces. Het moet letterlijk hardop denken, alsof ik je een vraag stelde en alleen een antwoord van je accepteerde in "stream of consciousness"-stijl.

Grote Taalmodellen Komen in Vele Maten en Smaken

Hoewel vrijwel alle populaire grote taalmodellen (LLMs) gebaseerd zijn op dezelfde kern-transformerarchitectuur en getraind zijn op enorme tekstdatasets, komen ze in verschillende groottes en zijn ze fijnafgestemd voor verschillende doeleinden. De grootte van een LLM, gemeten in het aantal parameters in zijn neuraal netwerk, heeft een grote invloed op zijn mogelijkheden. Grotere modellen met meer parameters, zoals GPT-4, waarvan wordt gezegd dat het 1 tot 2 biljoen parameters heeft, zijn over het algemeen meer kundig en bekwaam dan kleinere modellen. Grotere modellen hebben echter ook veel meer rekenkracht nodig om te draaien, wat zich vertaalt in hogere kosten wanneer je ze via API-aanroepen gebruikt.

Om LLMs praktischer en meer toegespitst te maken op specifieke gebruikssituaties, worden de basismodellen vaak fijnafgestemd op meer gerichte datasets. Zo kan een LLM worden getraind op een groot corpus van dialogen om het te specialiseren voor conversatie-AI. Andere worden getraind op code om ze te voorzien van programmeerkennis. Er zijn zelfs modellen die speciaal getraind zijn voor roleplay-stijl interacties met gebruikers!

Retrieval vs Generatieve Modellen

In de wereld van grote taalmodellen (LLMs) zijn er twee hoofdbenaderingen voor het genereren van responses: op retrieval gebaseerde modellen en generatieve modellen. Elke benadering heeft zijn eigen sterke en zwakke punten, en het begrijpen van de verschillen tussen beide kan je helpen het juiste model te kiezen voor jouw specifieke gebruikssituatie.

Op Retrieval Gebaseerde Modellen

Op retrieval gebaseerde modellen, ook bekend als informatie-ophalingsmodellen, genereren antwoorden door te zoeken in een grote database van bestaande tekst en de meest relevante passages te selecteren op basis van de invoerquery. Deze modellen genereren geen nieuwe tekst vanaf nul, maar voegen in plaats daarvan fragmenten uit de database samen om een samenhangend antwoord te vormen.

Een van de belangrijkste voordelen van op retrieval gebaseerde modellen is hun vermogen om feitelijk accurate en actuele informatie te verstrekken. Omdat ze vertrouwen op een database met gecureerde tekst, kunnen ze relevante informatie uit betrouwbare bronnen halen en deze aan de gebruiker presenteren. Dit maakt ze bijzonder geschikt voor toepassingen die precieze, feitelijke antwoorden vereisen, zoals vraag-antwoordsystemen of kennisbanken.

Deze op retrieval gebaseerde modellen hebben echter ook beperkingen. Ze zijn slechts zo goed als de database waarin ze zoeken, dus de kwaliteit en dekking van de database hebben directe invloed op de prestaties van het model. Daarnaast kunnen deze modellen moeite hebben met het genereren van samenhangende en natuurlijk klinkende antwoorden, omdat ze beperkt zijn tot de tekst die beschikbaar is in de database.

We behandelen het gebruik van pure retrievalmodellen niet in dit boek.

Generatieve Modellen

Generatieve modellen daarentegen creëren nieuwe tekst vanaf nul, gebaseerd op de patronen en relaties die ze tijdens de training hebben geleerd. Deze modellen gebruiken hun begrip van taal om nieuwe antwoorden te genereren die zijn toegespitst op de invoerprompt.

De belangrijkste kracht van generatieve modellen is hun vermogen om creatieve, samenhangende en contextueel relevante tekst te produceren. Ze kunnen open gesprekken voeren, verhalen genereren en zelfs code schrijven. Dit maakt ze ideaal voor toepassingen die meer open en dynamische interacties vereisen, zoals chatbots, contentcreatie en hulpmiddelen voor creatief schrijven.

Generatieve modellen kunnen echter soms inconsistente of feitelijk onjuiste informatie produceren, omdat ze vertrouwen op de patronen die tijdens de training zijn geleerd in plaats van een gecureerde database met feiten. Ze kunnen ook gevoeliger zijn voor vooroordelen en hallucinaties, waarbij ze tekst genereren die aannemelijk lijkt maar niet noodzakelijk waar is.

Voorbeelden van generatieve LLMs zijn OpenAI's GPT-serie (GPT-3, GPT-4) en Anthropic's Claude.

Hybride Modellen

Verschillende commercieel beschikbare LLMs combineren zowel retrieval als generatieve benaderingen in een hybride model. Deze modellen gebruiken retrievaltechnieken om relevante informatie uit een database te vinden en gebruiken vervolgens generatieve technieken om die informatie te synthetiseren tot een samenhangend antwoord.

Hybride modellen streven ernaar de feitelijke nauwkeurigheid van op retrieval gebaseerde modellen te combineren met de natuurlijke taalgeneratiecapaciteiten van

generatieve modellen. Ze kunnen betrouwbaardere en actuelere informatie verstrekken terwijl ze nog steeds in staat zijn om open gesprekken te voeren.

Bij het kiezen tussen op retrieval gebaseerde en generatieve modellen moet je rekening houden met de specifieke vereisten van je toepassing. Als het hoofddoel is om accurate, feitelijke informatie te verstrekken, kan een op retrieval gebaseerd model de beste keuze zijn. Als de toepassing meer open en creatieve interacties vereist, kan een generatief model geschikter zijn. Hybride modellen bieden een evenwicht tussen de twee benaderingen en kunnen een goede keuze zijn voor toepassingen die zowel feitelijke nauwkeurigheid als natuurlijke taalgeneratie vereisen.

Uiteindelijk hangt de keuze tussen op retrieval gebaseerde en generatieve modellen af van de specifieke gebruikssituatie en de afwegingen tussen nauwkeurigheid, creativiteit en flexibiliteit. Door de sterke punten en beperkingen van elke benadering te begrijpen, kun je weloverwogen beslissingen nemen.

Instructie-fijnafstemming

Instructie-fijnafstemming is een onderdeel van conversationele training dat gebruik maakt van door mensen geschreven prompts en antwoorden om LLMs beter te maken in het begrijpen en opvolgen van instructies. Hier zijn enkele eenvoudige voorbeelden om te illustreren hoe instructie-fijnafstemming op door mensen geschreven prompts en antwoorden LLMs beter maakt in het begrijpen en opvolgen van instructies:

Voorbeeld 1: Klantenondersteuning

Instructie: "Geef een antwoord aan een klant die om een terugbetaling vraagt."

Door Mensen Geschreven Prompt: "Ik wil graag een terugbetaling aanvragen voor mijn recente aankoop. Het product voldeed niet aan mijn verwachtingen."

Door Mensen Geschreven Antwoord: "Het spijt me te horen dat het product niet aan uw verwachtingen voldeed. Om uw terugbetaling te verwerken, geef dan alstublieft uw ordernummer en de reden voor retournering door. We zullen uw verzoek bekijken en binnen 2-3 werkdagen contact met u opnemen."

Voorbeeld 2: Educatieve Ondersteuning

Instructie: "Leg het concept van fotosynthese uit aan een 10-jarige."

Door mensen geschreven prompt: "Kun je uitleggen hoe planten hun voedsel maken?"

Door mensen geschreven antwoord: "Natuurlijk! Planten maken hun voedsel door een proces dat fotosynthese heet. Ze gebruiken zonlicht, water en koolstofdioxide om voedsel en zuurstof te maken. De bladeren van de plant hebben een speciaal onderdeel dat chlorofyl heet, dat hen helpt zonlicht op te vangen."

Voorbeeld 3: Persoonlijke Assistent

> Instructie: “Plan een vergadering voor volgende dinsdag om 15:00 uur.”
>
> Door mensen geschreven prompt: “Kun je een vergadering inplannen voor dinsdag om 15:00 uur?”
>
> Door mensen geschreven antwoord: “Natuurlijk, ik heb een vergadering voor je ingepland voor volgende dinsdag om 15:00 uur. Is er nog iets anders wat je nodig hebt?”

Het resultaat is een divers ecosysteem van LLM’s in verschillende groottes en met verschillende specialiteiten. Kleinere modellen in het bereik van 1-7 miljard parameters bieden goede algemene taalvaardigheden terwijl ze efficiënter zijn in gebruik.

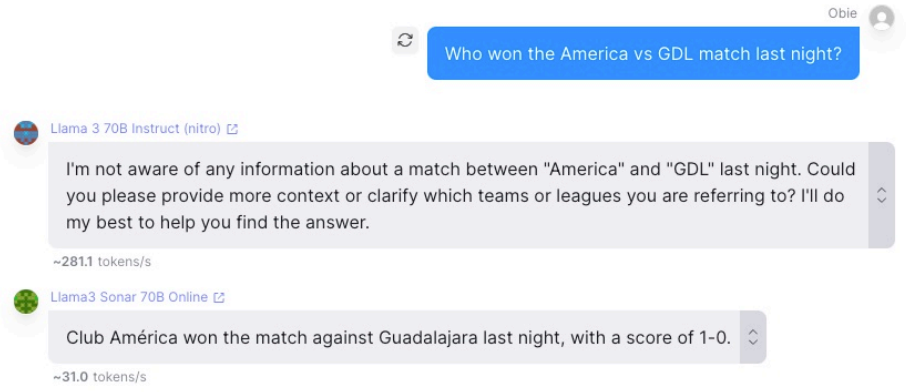
- Mistral 7B
- Llama 3 8B
- Gemma 7B

Middelgrote modellen van ongeveer 30-70 miljard parameters bieden sterkere redenerings- en instructievolgende vaardigheden.

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

Bij het kiezen van een LLM om in een applicatie te integreren, moet je de mogelijkheden van het model afwegen tegen praktische factoren zoals kosten, latentie, contextlengte en inhoudsfiltering. Kleinere, op instructies afgestemde modellen zijn vaak de beste keuze voor eenvoudigere taaltaken, terwijl de grootste modellen nodig kunnen zijn voor complexe redenering of analyse. De trainingsdata van het model is ook een belangrijke overweging, aangezien deze de kennisafkapdatum van het model bepaalt.

Bepaalde modellen, zoals sommige van Perplexity, zijn verbonden met realtime informatiebronnen, zodat ze effectief geen afkapdatum hebben. Wanneer je hen vragen stelt, kunnen ze zelfstandig beslissen om webzoekopdrachten uit te voeren en willekeurige webpagina's op te halen om een antwoord te genereren.

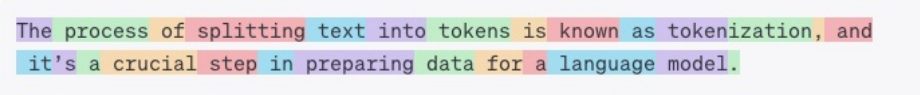


Figuur 1. Llama3 met en zonder online toegang

Uiteindelijk is er geen één-size-fits-all LLM. Inzicht in de variaties in modelgrootte, architectuur en training is essentieel voor het selecteren van het juiste model voor een specifieke toepassing. Experimenteren met verschillende modellen is de enige praktische manier om te ontdekken welke de beste prestaties leveren voor de taak in kwestie.

Tokenisatie: Tekst in stukken verdelen

Voordat een groot taalmodel tekst kan verwerken, moet die tekst worden opgedeeld in kleinere eenheden die *tokens* worden genoemd. Tokens kunnen individuele woorden, delen van woorden of zelfs enkele tekens zijn. Het proces van het opsplitsen van tekst in tokens wordt tokenisatie genoemd, en het is een cruciale stap in het voorbereiden van data voor een taalmodel.



Figuur 2. Deze zin bevat 27 tokens

Verschillende LLM's gebruiken verschillende tokenisatiestrategieën, die een significante impact kunnen hebben op de prestaties en mogelijkheden van het model. Enkele veelgebruikte tokenizers voor LLM's zijn:

- **GPT (Byte Pair Encoding)**: GPT-tokenizers gebruiken een techniek die byte pair encoding (BPE) wordt genoemd om tekst op te delen in subwoordeenheden. BPE voegt iteratief de meest voorkomende paren bytes in een tekstcorpus samen, waardoor een vocabulaire van subwoordtokens ontstaat. Dit stelt de tokenizer in staat om zeldzame en nieuwe woorden te verwerken door ze op te delen in meer voorkomende subwoorddelen. GPT-tokenizers worden gebruikt door modellen zoals GPT-3 en GPT-4.

- **Llama (SentencePiece)**: Llama-tokenizers gebruiken de SentencePiece-bibliotheek, een ongesuperviseerde teksttokenizer en detokenizer. SentencePiece behandelt de invoertekst als een reeks Unicode-tekens en leert een deelwoordvocabulaire op basis van een trainingscorpus. Het kan elke taal verwerken die in Unicode kan worden gecodeerd, waardoor het zeer geschikt is voor meertalige modellen. Llama-tokenizers worden gebruikt door modellen zoals Meta's Llama en Alpaca.

- **SentencePiece (Unigram)**: SentencePiece-tokenizers kunnen ook gebruik maken van een ander algoritme genaamd Unigram, dat gebaseerd is op een deelwoordregularisatietechniek. Unigram-tokenization bepaalt de optimale deelwoordvocabulaire op basis van een unigram-taalmodel, dat waarschijnlijkheden toekent aan individuele deelwoordeenheden. Deze aanpak kan semantisch betekenisvollere deelwoorden produceren in vergelijking met

BPE. SentencePiece met Unigram wordt gebruikt door modellen zoals Google's T5 en BERT.

- **Google Gemini (Multimodale Tokenization)**: Google Gemini gebruikt een tokenizatieschema dat is ontworpen om verschillende soorten gegevens te verwerken, waaronder tekst, afbeeldingen, audio, video's en code. Deze multimodale capaciteit stelt Gemini in staat om verschillende vormen van informatie te verwerken en te integreren. Met name Google Gemini 1.5 Pro heeft een contextvenster dat miljoenen tokens kan verwerken, veel meer dan eerdere modellen. Dit uitgebreide contextvenster stelt het model in staat om een grotere context te verwerken, wat mogelijk tot nauwkeurigere antwoorden leidt. Het is echter belangrijk op te merken dat Gemini's tokenizatieschema veel dichter bij één token per teken ligt dan andere modellen. Dit betekent dat de werkelijke kosten van het gebruik van Gemini-modellen aanzienlijk hoger kunnen zijn dan verwacht als je gewend bent aan het gebruik van modellen zoals GPT, aangezien Google's prijzen gebaseerd zijn op tekens in plaats van tokens.

De keuze van tokenizer beïnvloedt verschillende aspecten van een LLM, waaronder:

- **Vocabulairegrootte**: De tokenizer bepaalt de grootte van het vocabulaire van het model, wat de verzameling unieke tokens is die het herkent. Een groter, meer verfijnd vocabulaire kan het model helpen om een breder scala aan woorden en zinnen te verwerken en zelfs multimodaal te worden (in staat om meer dan alleen tekst te begrijpen en te genereren), maar het verhoogt ook de geheugenvereisten en computationele complexiteit van het model.
- **Omgang met zeldzame en onbekende woorden**: Tokenizers die deelwoordeenheden gebruiken, zoals BPE en SentencePiece, kunnen zeldzame en onbekende woorden opsplitsen in meer voorkomende deelwoorden. Dit stelt het model in staat om beredeneerde schattingen te maken over de betekenis van woorden die het nog niet eerder heeft gezien, gebaseerd op de deelwoorden waaruit ze bestaan.

- **Meertalige ondersteuning**: Tokenizers zoals SentencePiece, die elke in Unicode codeerbare taal kunnen verwerken, zijn zeer geschikt voor meertalige modellen die tekst in verschillende talen moeten verwerken.

Bij het kiezen van een LLM voor een specifieke toepassing is het belangrijk om rekening te houden met de gebruikte tokenizer en hoe goed deze aansluit bij de specifieke taalverwerkingsbehoeften van de taak. De tokenizer kan een significante impact hebben op het vermogen van het model om domeinspecifieke terminologie, zeldzame woorden en meertalige tekst te verwerken.

Contextgrootte: Hoeveel Informatie Kan een Taalmodel Gebruiken Tijdens Inferentie?

Bij het bespreken van taalmodellen verwijst contextgrootte naar de hoeveelheid tekst die een model kan overwegen bij het verwerken of genereren van zijn antwoorden. Het is in essentie een maat voor hoeveel informatie het model kan "onthouden" en gebruiken om zijn output te informeren (uitgedrukt in tokens). De contextgrootte van een taalmodel kan een significante impact hebben op zijn mogelijkheden en de soorten taken die het effectief kan uitvoeren.

Wat is Contextgrootte?

In technische termen wordt de contextgrootte bepaald door het aantal tokens (woorden of woorddelen) dat een taalmodel in één invoerreeks kan verwerken. Dit wordt vaak aangeduid als de "aandachtsspanne" of het "contextvenster" van het model. Hoe groter de contextgrootte, hoe meer tekst het model tegelijkertijd kan overwegen bij het genereren van een antwoord of het uitvoeren van een taak.

Verschillende taalmodellen hebben uiteenlopende contextgroottes, variërend van enkele honderden tokens tot miljoenen tokens. Ter referentie: een typische alinea tekst bevat ongeveer 100-150 tokens, terwijl een heel boek tienduizenden of honderdduizenden tokens kan bevatten.

> Er wordt zelfs gewerkt aan efficiënte methoden om Transformer-gebaseerde Large Language Models (LLMs) te schalen naar oneindig lange invoer met begrensde geheugen- en rekencapaciteit.

Waarom is Contextgrootte Belangrijk?

De contextgrootte van een taalmodel heeft een significante invloed op het vermogen om coherente, contextueel relevante tekst te begrijpen en te genereren. Hier zijn enkele belangrijke redenen waarom contextgrootte ertoe doet:

1. **Begrip van langere content**: Modellen met grotere contextgroottes kunnen langere teksten, zoals artikelen, rapporten of zelfs complete boeken, beter begrijpen en analyseren. Dit is cruciaal voor taken zoals documentsamenvattingen, het beantwoorden van vragen en inhoudsanalyse.

2. **Behoud van coherentie**: Een groter contextvenster stelt het model in staat om coherentie en consistentie te behouden over langere stukken output. Dit is belangrijk voor taken zoals het genereren van verhalen, dialoogsystemen en het creëren van content, waarbij het behouden van een consistente verhaallijn of onderwerp essentieel is. Het is ook absoluut cruciaal bij het gebruik van LLMs voor het genereren of transformeren van gestructureerde data.
3. **Vastleggen van langeafstandsafhankelijkheden**: Sommige taaltaken vereisen begrip van relaties tussen woorden of zinsdelen die ver uit elkaar staan in een tekst. Modellen met grotere contextgroottes zijn beter uitgerust om deze langeafstandsafhankelijkheden vast te leggen, wat belangrijk kan zijn voor taken zoals sentimentanalyse, vertaling, en taalbegrip.
4. **Omgaan met complexe instructies**: Bij toepassingen waar taalmodellen worden gebruikt om complexe, meerstaps instructies te volgen, zorgt een grotere

contextgrootte ervoor dat het model de volledige set instructies kan overwegen bij het genereren van een antwoord, in plaats van alleen de meest recente woorden.

Voorbeelden van Taalmodellen met Verschillende Contextgroottes

Hier zijn enkele voorbeelden van taalmodellen met verschillende contextgroottes:

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

Zoals je kunt zien, is er een breed scala aan contextgroottes onder deze modellen, van ongeveer 4.000 tokens voor het OpenAI GPT-3.5 Turbo model tot 200.000 tokens voor het Anthropic Claude v2 model. Sommige modellen, zoals Google's PaLM 2 en OpenAI's GPT-4, bieden verschillende varianten met grotere contextgroottes (bijvoorbeeld "32k" versies), die nog langere inputreeksen kunnen verwerken. En op dit moment (april 2024) pocht Google Gemini Pro met bijna 3 miljoen tokens!

Het is belangrijk op te merken dat de contextgrootte kan variëren afhankelijk van de specifieke implementatie en versie van een bepaald model. Zo heeft het originele OpenAI GPT-4 model een contextgrootte van 8.191 tokens, terwijl de latere GPT-4 varianten zoals Turbo en 4o een veel grotere contextgrootte van 128.000 tokens hebben.

> Sam Altman heeft de huidige contextbeperkingen vergeleken met de kilobytes aan werkgeheugen waarmee personal computer programmeurs in de jaren 80 moesten

> werken, en zei dat we in de nabije toekomst “al je persoonlijke data” in de context van een groot taalmodel zullen kunnen plaatsen.

De Juiste Contextgrootte Kiezen

Bij het selecteren van een taalmodel voor een specifieke toepassing is het belangrijk om rekening te houden met de contextgrootte-vereisten van de betreffende taak. Voor taken die korte, geïsoleerde tekstfragmenten betreffen, zoals sentimentanalyse of eenvoudige vraagbeantwoording, kan een kleinere contextgrootte voldoende zijn. Voor taken die echter begrip en generatie van langere, complexere teksten vereisen, zal een grotere contextgrootte waarschijnlijk noodzakelijk zijn.

Het is belangrijk op te merken dat grotere contextgroottes vaak gepaard gaan met hogere rekenkosten en langzamere verwerkingstijden, omdat het model meer informatie moet overwegen bij het genereren van een antwoord. Daarom moet je een balans vinden tussen contextgrootte en prestaties bij het kiezen van een taalmodel voor je toepassing.

> Waarom kiezen we niet gewoon het model met de grootste contextgrootte en stoppen we er zoveel mogelijk informatie in? Nou, naast prestatiefactoren is de andere belangrijke overweging de kosten. In maart 2024 kost een *enkele* prompt-responscyclus met Google Gemini Pro 1.5 met een volledige context je bijna $8 (USD). Als je een use case hebt die deze kosten rechtvaardigt, ga je gang! Maar voor de meeste toepassingen is het gewoonweg orders van grootte te duur.

Naalden Zoeken in Hooibergen

Het concept van een naald in een hooiberg zoeken is al lang een metafoor voor de uitdagingen van informatiewinning in grote datasets. Op het gebied van LLMs passen we deze analogie een beetje aan. Stel je voor dat we niet alleen op zoek zijn naar één feit dat verborgen ligt in een uitgebreide tekst (zoals een volledige verzameling essays van Paul Graham), maar naar meerdere feiten die overal verspreid liggen. Dit scenario lijkt meer op het zoeken naar verschillende naalden in een uitgestrekt veld, niet slechts één hooiberg. En hier komt het: we moeten deze naalden niet alleen vinden, maar ze ook tot een samenhangend geheel weven.

Wanneer LLMs de taak krijgen om meerdere feiten op te halen en daarover te redeneren binnen lange contexten, staan ze voor een dubbele uitdaging. Ten eerste is er het voor de hand liggende probleem van nauwkeurigheid bij het ophalen - die daalt natuurlijk naarmate het aantal feiten toeneemt. Dit is te verwachten; immers, het bijhouden van meerdere details in een uitgebreide tekst belast zelfs de meest geavanceerde modellen.

Ten tweede, en misschien wel belangrijker, is de uitdaging om met deze feiten te redeneren. Het is één ding om feiten te selecteren; het is iets heel anders om ze te synthetiseren tot een samenhangend verhaal of antwoord. Hier komt de echte test. De prestaties van LLMs bij redeneertaken nemen sterker af dan bij eenvoudige ophaalactiviteiten. Deze verslechtering gaat niet alleen over volume; het gaat om de ingewikkelde dans van context, relevantie en gevolgtrekking.

Waarom gebeurt dit? Welnu, kijk naar de dynamiek van geheugen en aandacht in menselijke cognitie, die tot op zekere hoogte wordt weerspiegeld in LLMs. Bij het verwerken van grote hoeveelheden informatie kunnen LLMs, net als mensen, eerdere details kwijtraken terwijl ze nieuwe opnemen. Dit geldt vooral voor modellen die niet expliciet zijn ontworpen om automatisch prioriteit te geven aan of terug te keren naar eerdere tekstsegmenten.

Bovendien is het vermogen van een LLM om deze opgehaalde feiten tot een samenhangend antwoord te weven vergelijkbaar met verhaalopbouw. Dit vereist niet

alleen het ophalen van informatie, maar ook een diep begrip en contextuele plaatsing, wat een grote uitdaging blijft voor de huidige AI.

Dus, wat betekent dit voor ons als ontwikkelaars en integreerders van deze technologieën? We moeten ons scherp bewust zijn van deze beperkingen bij het ontwerpen van systemen die vertrouwen op LLMs voor complexe, langdurige taken. Het begrip dat prestaties onder bepaalde omstandigheden kunnen verslechteren, helpt ons realistische verwachtingen te stellen en betere terugvalmechanismen of aanvullende strategieën te ontwikkelen.

Modaliteiten: Voorbij Tekst

Hoewel de meerderheid van de taalmodellen vandaag de dag gericht is op het verwerken en genereren van tekst, is er een groeiende trend naar multimodale modellen die van nature meerdere soorten gegevens kunnen invoeren en uitvoeren, zoals afbeeldingen, audio en video. Deze multimodale modellen openen nieuwe mogelijkheden voor AI-gestuurde toepassingen die inhoud over verschillende modaliteiten kunnen begrijpen en genereren.

Wat zijn Modaliteiten?

In de context van taalmodellen verwijzen modaliteiten naar de verschillende soorten gegevens die een model kan verwerken en genereren. De meest voorkomende modaliteit is tekst, waaronder geschreven taal in verschillende vormen zoals boeken, artikelen, websites en sociale media berichten. Er zijn echter verschillende andere modaliteiten die in toenemende mate worden opgenomen in taalmodellen:

- **Afbeeldingen**: Visuele gegevens zoals foto's, illustraties en diagrammen.
- **Audio**: Geluidsgegevens zoals spraak, muziek en omgevingsgeluiden.
- **Video**: Bewegende visuele gegevens, vaak vergezeld van audio, zoals videoclips en films.

Elke modaliteit brengt unieke uitdagingen en kansen met zich mee voor taalmodellen. Afbeeldingen vereisen bijvoorbeeld dat het model visuele concepten en relaties begrijpt, terwijl audio vereist dat het model spraak en andere geluiden verwerkt en genereert.

Multimodale Taalmodellen

Multimodale taalmodellen zijn ontworpen om meerdere modaliteiten binnen één model te verwerken. Deze modellen hebben meestal gespecialiseerde componenten of lagen die zowel invoer kunnen begrijpen als uitvoergegevens kunnen genereren in verschillende modaliteiten. Enkele opmerkelijke voorbeelden van multimodale taalmodellen zijn:

- **OpenAI's GPT-4o**: GPT-4o is een groot taalmodel dat van nature spraak-audio naast tekst begrijpt en verwerkt. Deze mogelijkheid stelt GPT-4o in staat om taken uit te voeren zoals het transcriberen van gesproken taal, het genereren van tekst uit audio-invoer en het geven van antwoorden op basis van gesproken vragen.
- **OpenAI's GPT-4 met visuele invoer**: GPT-4 is een groot taalmodel dat zowel tekst als afbeeldingen kan verwerken. Wanneer een afbeelding als invoer wordt gegeven, kan GPT-4 de inhoud van de afbeelding analyseren en tekst genereren die de visuele informatie beschrijft of daarop reageert.
- **Google's Gemini**: Gemini is een multimodaal model dat tekst, afbeeldingen en video kan verwerken. Het gebruikt een uniforme architectuur die cross-modale begrip en generatie mogelijk maakt, waardoor taken zoals beeldonderschriften genereren, video samenvatten en visuele vraagbeantwoording mogelijk worden.

- **DALL-E en Stable Diffusion**: Hoewel dit geen taalmodellen in de traditionele zin zijn, demonstreren deze modellen de kracht van multimodale AI door afbeeldingen te genereren uit tekstuele beschrijvingen. Ze tonen het potentieel van modellen die kunnen vertalen tussen verschillende modaliteiten.

Voordelen en Toepassingen van Multimodale Modellen

Multimodale taalmodellen bieden verschillende voordelen en maken een breed scala aan toepassingen mogelijk, waaronder:

- **Verbeterd begrip**: Door informatie uit meerdere modaliteiten te verwerken, kunnen deze modellen een uitgebreider begrip van de wereld krijgen, vergelijkbaar met hoe mensen leren van verschillende zintuiglijke inputs.
- **Crossmodale generatie**: Multimodale modellen kunnen inhoud in één modaliteit genereren op basis van input uit een andere modaliteit, zoals het creëren van een afbeelding uit een tekstbeschrijving of het genereren van een videosamenvatting uit een geschreven artikel.
- **Toegankelijkheid**: Multimodale modellen kunnen informatie toegankelijker maken door te vertalen tussen modaliteiten, zoals het genereren van tekstbeschrijvingen van afbeeldingen voor visueel beperkte gebruikers of het maken van audioversies van geschreven content.
- **Creatieve toepassingen**: Multimodale modellen kunnen worden gebruikt voor creatieve taken zoals het genereren van kunst, muziek of video's op basis van tekstuele prompts, wat nieuwe mogelijkheden opent voor kunstenaars en contentmakers.

Naarmate multimodale taalmodellen zich blijven ontwikkelen, zullen ze waarschijnlijk een steeds belangrijkere rol spelen in de ontwikkeling van AI-gestuurde toepassingen die inhoud over meerdere modaliteiten kunnen begrijpen en genereren. Dit zal leiden tot natuurlijkere en intuïtievere interacties tussen mensen en AI-systemen, en nieuwe mogelijkheden ontsluiten voor creatieve expressie en kennisverspreiding.

Provider-ecosystemen

Als het gaat om het integreren van grote taalmodellen (LLMs) in applicaties, is er een groeiend aantal opties om uit te kiezen. Elke grote LLM-provider, zoals OpenAI,

Anthropic, Google en Cohere, biedt zijn eigen ecosysteem van modellen, API's en tools. Bij het kiezen van de juiste provider moet rekening worden gehouden met verschillende factoren, waaronder prijzen, prestaties, inhoudsfiltering, gegevensprivacy en aanpassingsmogelijkheden.

OpenAI

OpenAI is een van de meest bekende providers van LLMs, waarbij de GPT-serie (GPT-3, GPT-4) breed wordt toegepast in verschillende applicaties. OpenAI biedt een gebruiksvriendelijke API waarmee je hun modellen eenvoudig kunt integreren in applicaties. Ze bieden een reeks modellen met verschillende mogelijkheden en prijspunten, van het instapmodel Ada tot het krachtige Davinci-model.

Het ecosysteem van OpenAI omvat ook tools zoals de OpenAI Playground, waarmee je kunt experimenteren met prompts en modellen kunt fijnafstemmen voor specifieke gebruikssituaties. Ze bieden inhoudsfilteringsopties om de generatie van ongepaste of schadelijke inhoud te voorkomen.

Bij het direct gebruiken van OpenAI's modellen vertrouw ik op Alex Rudall's ruby-openai bibliotheek.

Anthropic

Anthropic is een andere belangrijke speler in de LLM-ruimte, waarbij hun Claude-modellen aan populariteit winnen vanwege sterke prestaties en ethische overwegingen. Anthropic richt zich op het ontwikkelen van veilige en verantwoorde AI-systemen, met sterke nadruk op inhoudsfiltering en het vermijden van schadelijke outputs.

Het ecosysteem van Anthropic omvat de Claude API, waarmee je het model kunt integreren in hun applicaties, evenals tools voor promptengineering en fijnafstemming. Ze bieden ook het Claude Instant-model, dat websearchmogelijkheden integreert voor meer actuele en feitelijke antwoorden.

Bij het direct gebruiken van Anthropic's modellen vertrouw ik op Alex Rudall's anthrophic bibliotheek.

Google

Google heeft verschillende krachtige LLMs ontwikkeld, waaronder Gemini, BERT, T5 en PaLM. Deze modellen staan bekend om hun sterke prestaties op een breed scala aan natuurlijke taalverwerkingstaken. Het ecosysteem van Google omvat de TensorFlow- en Keras-bibliotheken, die tools en frameworks bieden voor het bouwen en trainen van machine learning-modellen.

Google biedt ook een Cloud AI-platform, waarmee je hun modellen eenvoudig kunt implementeren en schalen in de cloud. Ze bieden een reeks voorgetrainde modellen en API's voor taken zoals sentimentanalyse, entiteitsherkenning en vertaling.

Meta

Meta, voorheen bekend als Facebook, is diep geïnvesteerd in de ontwikkeling van grote taalmodellen, wat wordt benadrukt door de release van modellen zoals LLaMA en OPT. Deze modellen onderscheiden zich door hun sterke prestaties in diverse taaltaken en worden grotendeels beschikbaar gesteld via open-source kanalen, wat Meta's toewijding aan onderzoek en gemeenschapssamenwerking ondersteunt.

Het ecosysteem van Meta is voornamelijk gebouwd rond PyTorch, een open-source machine learning-bibliotheek die wordt gewaardeerd om zijn dynamische rekenmogelijkheden en flexibiliteit, wat innovatief AI-onderzoek en -ontwikkeling faciliteert.

Naast hun technische aanbod legt Meta sterk de nadruk op ethische AI-ontwikkeling. Ze implementeren robuuste inhoudsfiltering en richten zich op het verminderen van vooroordelen, in lijn met hun bredere doelstellingen van veiligheid en verantwoordelijkheid in AI-toepassingen.

Cohere

Cohere is een nieuwere speler in de LLM-ruimte, die zich richt op het toegankelijker en gebruiksvriendelijker maken van LLM's dan concurrenten. Hun ecosysteem omvat de Cohere API, die toegang biedt tot een reeks vooraf getrainde modellen voor taken zoals tekstgeneratie, classificatie en samenvatting.

Cohere biedt ook tools voor prompt engineering, fine-tuning en inhoudsfiltering. Ze leggen de nadruk op gegevensprivacy en beveiliging, met functies zoals versleutelde gegevensopslag en toegangscontrole.

Ollama

Ollama is een zelf-gehost platform waarmee gebruikers verschillende grote taalmodellen (LLM's) lokaal op hun machines kunnen beheren en implementeren, waardoor ze volledige controle hebben over hun AI-modellen zonder afhankelijk te zijn van externe clouddiensten. Deze opstelling is ideaal voor degenen die prioriteit geven aan gegevensprivacy en hun AI-operaties intern willen afhandelen.

Het platform ondersteunt een reeks modellen, waaronder versies van Llama, Phi, Gemma en Mistral, die verschillen in grootte en rekenvereisten. Ollama maakt het eenvoudig om deze modellen direct vanaf de opdrachtregel te downloaden en uit te voeren met eenvoudige commando's zoals `ollama run <model_name>`, en het is ontworpen om te werken op verschillende besturingssystemen, waaronder macOS, Linux en Windows.

Voor ontwikkelaars die open-source modellen in hun applicaties willen integreren zonder gebruik te maken van een externe API, biedt Ollama een CLI voor het beheren van modellevenscycli, vergelijkbaar met containerbeheertools. Het ondersteunt ook aangepaste configuraties en prompts, waardoor een hoge mate van aanpassing mogelijk is om de modellen af te stemmen op specifieke behoeften of gebruikssituaties.

Ollama is vooral geschikt voor technisch onderlegde gebruikers en ontwikkelaars

vanwege de opdrachtregelinterface en de flexibiliteit die het biedt bij het beheren en implementeren van AI-modellen. Dit maakt het een krachtig hulpmiddel voor bedrijven en individuen die robuuste AI-mogelijkheden nodig hebben zonder concessies te doen aan beveiliging en controle.

Multi-Model Platforms

Daarnaast zijn er aanbieders die een breed scala aan open-source modellen hosten, zoals Together.ai en Groq. Deze platforms bieden flexibiliteit en aanpassingsmogelijkheden, waardoor je open-source modellen kunt draaien en in sommige gevallen zelfs kunt fine-tunen volgens jouw specifieke behoeften. Together.ai biedt bijvoorbeeld toegang tot een reeks open-source LLM's, waardoor gebruikers kunnen experimenteren met verschillende modellen en configuraties. Groq richt zich op het leveren van ultrahoogwaardige voltooiing die op het moment van schrijven van dit boek bijna magisch lijkt

Een LLM Provider Kiezen

Bij het kiezen van een LLM-provider moet je rekening houden met factoren zoals:

- **Prijzen**: Verschillende providers bieden verschillende prijsmodellen, variërend van betalen naar gebruik tot abonnementsgebaseerde plannen. Het is belangrijk om rekening te houden met het verwachte gebruik en budget bij het selecteren van een provider.
- **Prestaties**: De prestaties van LLM's kunnen aanzienlijk verschillen tussen providers, dus het is belangrijk om modellen te benchmarken en te testen op specifieke gebruikssituaties voordat er een beslissing wordt genomen.
- **Inhoudsfiltering**: Afhankelijk van de toepassing kan inhoudsfiltering een cruciale overweging zijn. Sommige providers bieden robuustere inhoudsfilteringsopties dan andere.

- **Gegevensprivacy**: Als de toepassing gevoelige gebruikersgegevens verwerkt, is het belangrijk om een provider te kiezen met sterke praktijken op het gebied van gegevensprivacy en beveiliging.
- **Aanpassing**: Sommige providers bieden meer flexibiliteit wat betreft fine-tuning en het aanpassen van modellen voor specifieke gebruikssituaties.

Uiteindelijk hangt de keuze van LLM-provider af van de specifieke vereisten en beperkingen van de toepassing. Door zorgvuldig de opties te evalueren en rekening te houden met factoren zoals prijzen, prestaties en gegevensprivacy, kun je de provider selecteren die het beste aan je behoeften voldoet.

Het is ook vermeldenswaardig dat het LLM-landschap constant evolueert, met regelmatig nieuwe providers en modellen die verschijnen. Je moet op de hoogte blijven van de laatste ontwikkelingen en openstaan voor het verkennen van nieuwe opties wanneer deze beschikbaar komen.

OpenRouter

In dit boek zal ik uitsluitend gebruik maken van OpenRouter als mijn API-provider van keuze. De reden is eenvoudig: het is een one-stop shop voor alle meest populaire commerciële en open-source modellen. Als je staat te popelen om aan de slag te gaan met AI-programmeren, is een van de beste plekken om te beginnen mijn eigen OpenRouter Ruby Library.

Nadenken over Prestaties

Bij het integreren van taalmodellen in applicaties is prestatie een cruciale overweging. De prestatie van een taalmodel kan worden gemeten in termen van *latentie* (de tijd die nodig is om een reactie te genereren) en *doorvoer* (het aantal verzoeken dat per tijdseenheid kan worden verwerkt).

Tijd tot Eerste Token (TTET) is nog een essentiële prestatiemeting, die vooral relevant is voor chatbots en applicaties die interactieve, realtime reacties vereisen. TTET meet de latentie vanaf het moment dat het verzoek van een gebruiker wordt ontvangen tot het moment dat het eerste woord (of token) van het antwoord wordt gegenereerd. Deze meting is cruciaal voor het behouden van een soepele en boeiende gebruikerservaring, aangezien vertraagde reacties kunnen leiden tot frustratie en verminderde betrokkenheid van gebruikers.

Deze prestatiemetingen kunnen een significante impact hebben op de gebruikerservaring en de schaalbaarheid van de applicatie.

Verschillende factoren kunnen de prestaties van een taalmodel beïnvloeden, waaronder:

Parameteraantal: Grotere modellen met meer parameters hebben doorgaans meer computationele middelen nodig en kunnen een hogere latentie en lagere doorvoer hebben in vergelijking met kleinere modellen.

Hardware: De prestaties van een taalmodel kunnen aanzienlijk variëren afhankelijk van de hardware waarop het draait. Cloudproviders bieden GPU- en TPU-instances aan die geoptimaliseerd zijn voor machine learning-workloads, wat de modelinferentie aanzienlijk kan versnellen.

Een van de fijne dingen aan OpenRouter is dat je voor veel van de aangeboden modellen kunt kiezen uit verschillende cloudproviders met uiteenlopende prestatieprofielen en kosten.

Kwantisatie: Kwantisatietechnieken kunnen worden gebruikt om de geheugenvoetafdruk en computationele vereisten van een model te verminderen door gewichten en activaties met lagere precisie datatypen weer te geven. Dit kan de prestaties verbeteren zonder significant kwaliteitsverlies. Als applicatieontwikkelaar zul je waarschijnlijk niet betrokken zijn bij het trainen van je eigen modellen op verschillende kwantisatieniveaus, maar het is goed om ten minste bekend te zijn met de terminologie.

Batchverwerking: Het gelijktijdig verwerken van meerdere verzoeken in batches kan de doorvoer verbeteren door de overhead van het laden van modellen en gegevensoverdracht te spreiden.

Caching: Het cachen van resultaten van veelgebruikte prompts of invoersequenties kan het aantal inferentieverzoeken verminderen en de algehele prestaties verbeteren.

Bij het selecteren van een taalmodel voor een productieapplicatie is het belangrijk om de prestaties te benchmarken op representatieve workloads en hardwareconfiguraties. Dit kan helpen bij het identificeren van potentiële knelpunten en zorgen dat het model aan de vereiste prestatiedoelen kan voldoen.

Het is ook de moeite waard om de afwegingen tussen modelprestaties en andere factoren zoals kosten, flexibiliteit en integratiegemak te overwegen. Bijvoorbeeld, het gebruik van een kleiner, goedkoper model met lagere latentie kan de voorkeur hebben voor applicaties die realtime reacties vereisen, terwijl een groter, krachtiger model beter geschikt kan zijn voor batchverwerking of complexe redeneertaken.

Experimenteren met Verschillende GTM-modellen

Het kiezen van een GTM is zelden een permanente beslissing. Aangezien er regelmatig nieuwe en verbeterde modellen worden uitgebracht, is het goed om applicaties op een modulaire manier te bouwen die het mogelijk maakt om in de loop van de tijd verschillende taalmodellen uit te wisselen. Prompts en datasets kunnen vaak met minimale aanpassingen worden hergebruikt tussen modellen. Dit stelt je in staat om te profiteren van de nieuwste ontwikkelingen in taalmodellering zonder de applicaties volledig te hoeven herontwerpen.

De mogelijkheid om eenvoudig tussen een breed scala aan modelkeuzes te wisselen is nog een reden waarom ik dol ben op OpenRouter.

Bij het upgraden naar een nieuw taalmodel is het belangrijk om de prestaties en outputkwaliteit grondig te testen en te valideren om er zeker van te zijn dat het aan de vereisten van de applicatie voldoet. Dit kan het hertrainen of fine-tunen van het model op domeinspecifieke data omvatten, evenals het updaten van downstream componenten die afhankelijk zijn van de outputs van het model.

Door applicaties te ontwerpen met prestaties en modulariteit in gedachten, kun je schaalbare, efficiënte en toekomstbestendige systemen creëren die zich kunnen aanpassen aan het snel ontwikkelende landschap van taalmodeltechnologie.

Samengestelde AI-systemen

Voordat we onze introductie afsluiten, is het vermeldenswaardig dat vóór 2023 en de explosie van interesse in generatieve AI, aangewakkerd door ChatGPT, traditionele AI-benaderingen meestal vertrouwden op de integratie van enkele, gesloten modellen. Daarentegen maken *Samengestelde AI-systemen* gebruik van complexe pijplijnen van onderling verbonden componenten die samenwerken om intelligent gedrag te bereiken.

In de kern bestaan samengestelde AI-systemen uit meerdere modules, elk ontworpen om specifieke taken of functies uit te voeren. Deze modules kunnen generators, retrievers, rankers, classificatiesystemen en verschillende andere gespecialiseerde componenten bevatten. Door het algehele systeem op te delen in kleinere, gerichte eenheden kunnen ontwikkelaars flexibelere, schaalbaarder en beter onderhoudbare AI-architecturen creëren.

Een van de belangrijkste voordelen van samengestelde AI-systemen is hun vermogen om de sterke punten van verschillende AI-technieken en modellen te combineren. Een systeem kan bijvoorbeeld een groot taalmodel (LLM) gebruiken voor natuurlijke taalverwerking en -generatie, terwijl het een apart model inzet voor informatieopvraging of regelgebaseerde besluitvorming. Deze modulaire aanpak stelt je in staat om de beste hulpmiddelen en technieken voor elke specifieke taak te selecteren, in plaats van te vertrouwen op een one-size-fits-all oplossing.

Het bouwen van samengestelde AI-systemen brengt echter ook unieke uitdagingen met zich mee. Met name het waarborgen van de algehele samenhang en consistentie van het systeemgedrag vereist robuuste test-, monitoring- en besturingsmechanismen.

De komst van krachtige LLM's zoals GPT-4 stelt ons in staat gemakkelijker dan ooit te experimenteren met samengestelde AI-systemen, omdat deze geavanceerde modellen in staat zijn meerdere rollen binnen een samengesteld systeem te vervullen, zoals classificatie, rangschikking en generatie, naast hun natuurlijke taalverwerkingscapaciteiten. Deze veelzijdigheid stelt ontwikkelaars in staat om snel prototypes te maken en te itereren op samengestelde AI-architecturen, waardoor nieuwe mogelijkheden ontstaan voor de ontwikkeling van intelligente toepassingen.

Implementatiepatronen voor Samengestelde AI-systemen

Samengestelde AI-systemen kunnen worden geïmplementeerd met verschillende patronen, elk ontworpen om aan specifieke vereisten en gebruiksscenario's te voldoen. Laten we vier veel voorkomende implementatiepatronen verkennen: Vraag en Antwoord, Multi-Agent/Agentische Probleemoplossers, Conversationele AI, en CoPilots.

Vraag en Antwoord

Vraag en Antwoord (V&A) systemen richten zich op het leveren van informatieopvraging die wordt versterkt met de begripscapaciteiten van AI-modellen om meer te functioneren dan alleen als zoekmachine. Door krachtige taalmodellen te combineren met externe kennisbronnen met behulp van Retrieval-Augmented Generation (RAG), vermijden Vraag en Antwoord systemen hallucinaties en geven ze nauwkeurige en contextueel relevante antwoorden op gebruikersvragen.

De belangrijkste componenten van een LLM-gebaseerd V&A-systeem zijn:

- **Query-begrip en -herformulering**: Het analyseren van gebruikersvragen en deze herformuleren om beter aan te sluiten bij de onderliggende kennisbronnen.
- **Kennisopvraging**: Het ophalen van relevante informatie uit gestructureerde of ongestructureerde gegevensbronnen op basis van de geherformuleerde query.
- **Antwoordgeneratie**: Het genereren van samenhangende en informatieve antwoorden door de opgehaalde kennis te integreren met de generatieve mogelijkheden van het taalmodel.

RAG-subsystemen zijn vooral belangrijk in V&A-domeinen waar het verstrekken van nauwkeurige en actuele informatie cruciaal is, zoals klantenondersteuning, kennisbeheer, of educatieve toepassingen.

Multi-Agent/Agentische Probleemoplossers

Multi-agent, ook bekend als *Agentische*, systemen bestaan uit meerdere autonome agenten die samenwerken om complexe problemen op te lossen. Elke agent heeft een specifieke rol, set vaardigheden en toegang tot relevante hulpmiddelen of informatiebronnen. Door samen te werken en informatie uit te wisselen, kunnen deze agenten taken aanpakken die voor één enkele agent moeilijk of onmogelijk te hanteren zouden zijn.

De belangrijkste principes van multi-agent probleemoplossers zijn:

- **Specialisatie**: Elke agent richt zich op een specifiek aspect van het probleem, gebruikmakend van zijn unieke capaciteiten en kennis.
- **Samenwerking**: Agenten communiceren en coördineren hun acties om een gemeenschappelijk doel te bereiken, vaak door middel van het doorgeven van berichten of gedeeld geheugen.
- **Aanpasbaarheid**: Het systeem kan zich aanpassen aan veranderende omstandigheden of vereisten door de rollen en gedragingen van individuele agenten aan te passen.

Multi-agent systemen zijn zeer geschikt voor toepassingen die gedistribueerde probleemoplossing vereisen, zoals supply chain optimalisatie, verkeersbeheer, of planning van noodhulp.

Conversationele AI

Conversationele AI-systemen maken natuurlijke taalinteracties mogelijk tussen gebruikers en intelligente agenten. Deze systemen combineren natuurlijke taalverwerking, dialoogbeheer en taalgeneratiecapaciteiten om boeiende en gepersonaliseerde gesprekservaringen te bieden.

De belangrijkste componenten van een conversationeel AI-systeem zijn:

- **Intentieherkenning**: Het identificeren van de intentie van de gebruiker op basis van hun input, zoals het stellen van een vraag, het doen van een verzoek of het uiten van een sentiment.
- **Entiteitsextractie**: Het extraheren van relevante entiteiten of parameters uit de input van de gebruiker, zoals datums, locaties of productnamen.
- **Dialoogbeheer**: Het bijhouden van de staat van het gesprek, het bepalen van het juiste antwoord op basis van de intentie en context van de gebruiker, en het afhandelen van meerstaps-interacties.
- **Antwoordgeneratie**: Het genereren van mensachtige antwoorden met behulp van taalmodellen, sjablonen of op opvraging gebaseerde methoden.

Conversationele AI-systemen worden veel gebruikt in klantenservice chatbots, virtuele assistenten, en spraakgestuurde interfaces. Zoals eerder vermeld, zijn de meeste benaderingen, patronen en codevoorbeelden in dit boek direct afkomstig uit mijn werk aan een groot conversationeel AI-systeem genaamd Olympia.

CoPilots

CoPilots zijn AI-aangedreven assistenten die samenwerken met menselijke gebruikers om hun productiviteit en besluitvorming te verbeteren. Deze systemen maken

gebruik van een combinatie van natuurlijke taalverwerking, machine learning en domeinspecifieke kennis om intelligente aanbevelingen te doen, taken te automatiseren en contextuele ondersteuning te bieden.

Belangrijke kenmerken van CoPilots zijn:

- **Personalisatie**: Aanpassing aan individuele gebruikersvoorkeuren, werkstromen en communicatiestijlen.
- **Proactieve assistentie**: Anticiperen op gebruikersbehoeften en relevante suggesties of acties aanbieden zonder expliciete opdrachten.
- **Continue ontwikkeling**: Prestatieverbetering door te leren van gebruikersfeedback, interacties en gegevens.

CoPilots worden in toenemende mate gebruikt in verschillende domeinen, zoals softwareontwikkeling (bijvoorbeeld codecompletering en foutdetectie), creatief schrijven (bijvoorbeeld contentvoorstellen en bewerking), en data-analyse (bijvoorbeeld inzichten en visualisatie-aanbevelingen)

Deze implementatiepatronen tonen de veelzijdigheid en het potentieel van samengestelde AI-systemen aan. Door de kenmerken en gebruikssituaties van elk patroon te begrijpen, kunt u weloverwogen beslissingen nemen bij het ontwerpen en implementeren van intelligente applicaties. Hoewel dit boek niet specifiek gaat over de implementatie van samengestelde AI-systemen, zijn veel, zo niet alle, van dezelfde benaderingen en patronen van toepassing op het integreren van afzonderlijke AI-componenten binnen verder traditionele applicatieontwikkeling.

Rollen in Samengestelde AI-systemen

Samengestelde AI-systemen zijn gebouwd op een fundament van onderling verbonden modules, elk ontworpen om een specifieke rol te vervullen. Deze modules werken samen om intelligent gedrag te creëren en complexe problemen op te lossen. Het is nuttig om bekend te zijn met deze rollen wanneer u nadenkt over waar u mogelijk delen van uw applicatie kunt implementeren of vervangen door afzonderlijke AI-componenten.

Generator

Generators zijn verantwoordelijk voor het produceren van nieuwe gegevens of content op basis van geleerde patronen of inputprompts. De AI-wereld kent vele verschillende soorten generators, maar in de context van de taalmodellen die in dit boek worden getoond, kunnen generators mensachtige tekst creëren, onvolledige zinnen aanvullen of antwoorden genereren op gebruikersvragen. Ze spelen een cruciale rol bij taken zoals contentcreatie, dialooggeneratie en data-augmentatie.

Retriever

Retrievers worden gebruikt om relevante informatie te zoeken en te extraheren uit grote datasets of kennisbanken. Ze gebruiken technieken zoals semantisch zoeken, trefwoordovereenkomst of vectorgelijkenis om de meest relevante datapunten te vinden op basis van een gegeven zoekopdracht of context. Retrievers zijn essentieel voor taken die snelle toegang tot specifieke informatie vereisen, zoals het beantwoorden van vragen, feitencontrole of contentaanbevelingen.

Ranker

Rankers zijn verantwoordelijk voor het ordenen of prioriteren van een reeks items op basis van bepaalde criteria of relevantiescores. Ze kennen gewichten of scores toe aan elk item en sorteren ze dienovereenkomstig. Rankers worden vaak gebruikt in zoekmachines, aanbevelingssystemen of elke applicatie waarbij het presenteren van de meest relevante resultaten aan gebruikers cruciaal is.

Classifier

Classifiers worden gebruikt om datapunten te categoriseren of te labelen op basis van voorgedefinieerde klassen of categorieën. Ze leren van gelabelde trainingsgegevens en voorspellen vervolgens de klasse van nieuwe, onbekende instanties. Classifiers zijn

fundamenteel voor taken zoals sentimentanalyse, spamdetectie of beeldherkenning, waarbij het doel is om een specifieke categorie toe te wijzen aan elke input.

Tools & Agents

Naast deze kernrollen integreren samengestelde AI-systemen vaak tools en agents om hun functionaliteit en aanpassingsvermogen te verbeteren:

- **Tools**: Tools zijn afzonderlijke softwarecomponenten of API's die specifieke acties of berekeningen uitvoeren. Ze kunnen worden aangeroepen door andere modules, zoals generators of retrievers, om deeltaken uit te voeren of aanvullende informatie te verzamelen. Voorbeelden van tools zijn zoekmachines, rekenmachines of datavisualisatiebibliotheken.
- **Agents**: Agents zijn autonome entiteiten die hun omgeving kunnen waarnemen, beslissingen kunnen nemen en acties kunnen ondernemen om specifieke doelen te bereiken. Ze maken vaak gebruik van een combinatie van verschillende AI-technieken, zoals planning, redenering en leren, om effectief te functioneren in dynamische of onzekere omstandigheden. Agents kunnen worden gebruikt om complex gedrag te modelleren of om de acties van meerdere modules binnen een samengesteld AI-systeem te coördineren.

In een puur samengesteld AI-systeem wordt de interactie tussen deze componenten georkestreerd via welgedefinieerde interfaces en communicatieprotocollen. Gegevens stromen tussen modules, waarbij de output van de ene component dient als input voor de andere. Deze modulaire architectuur zorgt voor flexibiliteit, schaalbaarheid en onderhoudbaarheid, aangezien individuele componenten kunnen worden bijgewerkt, vervangen of uitgebreid zonder het hele systeem te beïnvloeden.

Door gebruik te maken van de kracht van deze componenten en hun interacties kunnen samengestelde AI-systemen complexe, praktische problemen aanpakken die een combinatie van verschillende AI-mogelijkheden vereisen. Terwijl we de benaderingen

en patronen voor het integreren van AI in applicatieontwikkeling verkennen, houd in gedachten dat dezelfde principes en technieken die worden gebruikt in samengestelde AI-systemen kunnen worden toegepast om intelligente, adaptieve en gebruikersgerichte applicaties te creëren.

In de volgende hoofdstukken van Deel 1 zullen we dieper ingaan op de fundamentele benaderingen en technieken voor het integreren van AI-componenten in uw applicatieontwikkelingsproces. Van prompt engineering en retrieval-augmented generation tot zelfherstellende data en intelligente workflow-orkestratie, we zullen een breed scala aan patronen en best practices behandelen om u te helpen geavanceerde AI-aangedreven applicaties te bouwen.

Deel 1: Fundamentele Benaderingen & Technieken

Dit deel van het boek presenteert verschillende manieren om het gebruik van AI in je applicaties te integreren. De hoofdstukken behandelen een reeks verwante benaderingen en technieken, variërend van meer abstracte concepten zoals Het Pad Versmallen en Retrieval Augmented Generation tot aan ideeën voor het programmeren van je eigen abstractielaag bovenop LLM chat completion APIs.

Het doel van dit deel van het boek is om je te helpen begrijpen welke soorten gedrag je kunt implementeren met AI, voordat we te diep ingaan op specifieke implementatiepatronen die de focus zijn van Deel 2.

De benaderingen in Deel 1 zijn gebaseerd op ideeën die ik in mijn code heb gebruikt, klassieke patronen van enterprise applicatie architectuur en integratie, plus metaforen die ik heb gebruikt bij het uitleggen van de mogelijkheden van AI aan andere mensen, waaronder niet-technische zakelijke belanghebbenden.

Het Pad Vernauwen

“Het pad vernauwen” verwijst naar het focussen van de AI op de huidige taak. Ik gebruik het als een mantra wanneer ik gefrustreerd raak omdat de AI zich “dom” gedraagt of onverwachte dingen doet. Het mantra herinnert me eraan dat het falen waarschijnlijk mijn schuld is, en dat ik het pad waarschijnlijk nog meer moet vernauwen.

De noodzaak om het pad te vernauwen komt voort uit de enorme hoeveelheid kennis die grote taalmodellen bevatten, vooral wereldklasse modellen zoals die van OpenAI en Anthropic die letterlijk biljoenen parameters hebben.

Toegang hebben tot zo'n breed scala aan kennis is ongetwijfeld krachtig en produceert emergent gedrag zoals theory of mind en het vermogen om op mensachtige wijze te redeneren. Deze overweldigende hoeveelheid informatie brengt echter ook uitdagingen met zich mee als het gaat om het genereren van precieze en accurate antwoorden op specifieke prompts, vooral als deze prompts bedoeld zijn om deterministisch gedrag te vertonen dat kan worden geïntegreerd met "normale" softwareontwikkeling en algoritmes.

Een aantal factoren leidt tot deze uitdagingen.

Informatie-overload: Grote taalmodellen worden getraind op enorme hoeveelheden data uit verschillende domeinen, bronnen en tijdsperioden. Deze uitgebreide kennis stelt hen in staat om deel te nemen aan diverse onderwerpen en antwoorden te genereren op basis van een breed begrip van de wereld. Echter, wanneer geconfronteerd met een specifieke prompt, kan het model moeite hebben om irrelevante, tegenstrijdige of verouderde/achterhaalde informatie te filteren, wat leidt tot antwoorden die focus of nauwkeurigheid missen. Afhankelijk van wat je probeert te doen, kan de pure hoeveelheid *tegenstrijdige* informatie die beschikbaar is voor het model gemakkelijk zijn vermogen overweldigen om het antwoord of gedrag te leveren dat je zoekt.

Contextuele Ambiguïteit: Gezien de enorme *latente ruimte* aan kennis, kunnen grote taalmodellen ambiguïteit tegenkomen bij het proberen te begrijpen van de *context* van je prompt. Zonder goede vernauwing of begeleiding kan het model antwoorden genereren die zijdelings gerelateerd zijn maar niet direct relevant voor je bedoelingen. Dit soort falen leidt tot antwoorden die niet ter zake doen, inconsistent zijn, of niet aan je gestelde behoeften voldoen. In dit geval verwijst het vernauwen van het pad naar context *disambiguatie*, waarbij wordt verzekerd dat de context die je biedt het model alleen laat focussen op de meest relevante informatie in zijn basiskennis.

Opmerking: Als je net begint met "prompt engineering" is de kans veel groter dat je het model dingen vraagt zonder het gewenste resultaat goed uit te leggen; het vergt oefening om niet ambigu te zijn!

Temporele Inconsistenties: Omdat taalmodellen zijn getraind op data die in verschillende tijdsperioden is gecreëerd, kunnen ze kennis bezitten die verouderd, achterhaald of niet meer accuraat is. Bijvoorbeeld, informatie over actuele gebeurtenissen, wetenschappelijke ontdekkingen of technologische vooruitgang kan zijn geëvolueerd sinds de trainingsdata van het model werd verzameld. Zonder het pad te vernauwen om prioriteit te geven aan recentere en betrouwbaardere bronnen, kan het model antwoorden genereren op basis van verouderde of onjuiste informatie, wat leidt tot onnauwkeurigheden en inconsistenties in zijn output.

Domeinspecifieke Nuances: Verschillende domeinen en vakgebieden hebben hun eigen specifieke terminologie, conventies en kennisbasissen. Denk aan vrijwel elke TLA (Three Letter Acronym) en je zult beseffen dat de meeste meer dan één betekenis hebben. MSK kan bijvoorbeeld verwijzen naar Amazon's Managed Streaming for Apache Kafka, het Memorial Sloan Kettering Cancer Center, of het menselijke MusculoSKeletale systeem.

Wanneer een prompt expertise in een bepaald domein vereist, is de algemene kennis van een groot taalmodel mogelijk niet voldoende om accurate en genuanceerde antwoorden te geven. Het vernauwen van het pad door te focussen op domeinspecifieke informatie, hetzij door prompt engineering of retrieval-augmented generation, stelt het model in staat om antwoorden te genereren die beter aansluiten bij de vereisten en verwachtingen van je specifieke domein.

Latente Ruimte: Onbegrijpelijk Uitgestrekt

Wanneer ik de "latente ruimte" van een taalmodel noem, verwijs ik naar het uitgestrekte, multidimensionale landschap van kennis en informatie dat het model heeft geleerd tijdens zijn trainingsproces. Het is als een verborgen rijk binnen de neurale netwerken van het model, waar alle patronen, associaties en representaties van taal zijn opgeslagen.

Stel je voor dat je een uitgestrekt, onontdekt gebied verkent vol met ontelbare onderling verbonden knooppunten. Elk knooppunt vertegenwoordigt een stukje informatie, een

concept of een relatie die het model heeft geleerd. Terwijl je door deze ruimte navigeert, zul je merken dat sommige knooppunten dichter bij elkaar liggen, wat duidt op een sterke connectie of gelijkenis, terwijl andere verder uit elkaar liggen, wat een zwakkere of meer afstandelijke relatie suggereert.

De uitdaging met de latente ruimte is dat deze ongelooflijk complex en hoogdimensionaal is. Zie het als iets zo immens als ons fysieke universum, met zijn clusters van sterrenstelsels en de enorme, onvoorstelbare afstanden van lege ruimte ertussen.

Omdat het duizenden dimensies bevat, is de latente ruimte niet direct waarneembaar of interpreteerbaar door mensen. Het is een abstracte representatie die het model intern gebruikt om taal te verwerken en te genereren. Wanneer je een input prompt aan het model geeft, brengt het deze prompt in wezen in kaart op een specifieke locatie binnen de latente ruimte. Het model gebruikt vervolgens de omringende informatie en verbindingen in die ruimte om een antwoord te genereren.

Het punt is dat het model een enorme hoeveelheid informatie heeft geleerd van zijn trainingsgegevens, en niet alles daarvan is relevant of accuraat voor een bepaalde taak. Daarom wordt het versmallen van het pad zo belangrijk. Door duidelijke instructies, voorbeelden en context in je prompts te geven, stuur je het model in feite naar specifieke regio's binnen de latente ruimte die het meest relevant zijn voor je gewenste output.

Een andere manier om erover na te denken is als het gebruiken van een spotlight in een volledig donker museum. Als je ooit het Louvre of Metropolitan Museum of Art hebt bezocht, dan is dat de schaal waarover ik het heb. De latente ruimte is het museum, gevuld met ontelbare objecten en details. Je prompt is de spotlight die specifieke gebieden verlicht en de aandacht van het model vestigt op de belangrijkste informatie. Zonder die sturing kan het model doelloos door de latente ruimte dwalen en onderweg irrelevante of tegenstrijdige informatie oppikken.

Terwijl je met taalmodellen werkt en je prompts opstelt, houd dan het concept van latente ruimte in gedachten. Je doel is om effectief door dit uitgestrekte kennislandschap

te navigeren en het model naar de meest relevante en accurate informatie voor je taak te sturen. Door het pad te versmallen en duidelijke sturing te geven, kun je het volledige potentieel van de latente ruimte van het model ontsluiten en kwalitatief hoogwaardige, coherente antwoorden genereren.

Hoewel de voorgaande beschrijvingen van taalmodellen en de latente ruimte waarin ze navigeren misschien wat magisch of abstract lijken, is het belangrijk te begrijpen dat prompts geen toverspreuken of bezweringen zijn. De manier waarop taalmodellen werken is gebaseerd op de principes van lineaire algebra en waarschijnlijkheidstheorie.

In essentie zijn taalmodellen probabilistische modellen van tekst, vergelijkbaar met hoe een klokcurve een statistisch model van data is. Ze worden getraind via een proces dat autoregressieve modellering wordt genoemd, waarbij het model leert om de waarschijnlijkheid van het volgende woord in een reeks te voorspellen op basis van de woorden die eraan voorafgaan. Tijdens de training begint het model met willekeurige gewichten en past deze geleidelijk aan om hogere waarschijnlijkheden toe te kennen aan tekst die lijkt op de praktijkvoorbeelden waarop het werd getraind.

Echter, het beschouwen van taalmodellen als eenvoudige statistische modellen, zoals lineaire regressie, biedt niet de beste intuïtie voor het begrijpen van hun gedrag. Een betere analogie is om ze te zien als probabilistische programma's, wat modellen zijn die de manipulatie van willekeurige variabelen mogelijk maken en complexe statistische relaties kunnen weergeven.

Probabilistische programma's kunnen worden weergegeven door grafische modellen, die een visuele manier bieden om de afhankelijkheden en relaties tussen variabelen in het model te begrijpen. Dit perspectief kan waardevolle inzichten bieden in de werking van complexe tekstgeneratiemodellen zoals GPT-4 en Claude.

In het artikel "Language Model Cascades" van Dohan et al. duiken de auteurs in de details van hoe probabilistische programma's kunnen worden toegepast op taalmodellen. Ze laten zien hoe dit raamwerk kan worden gebruikt om het gedrag van deze modellen te begrijpen en de ontwikkeling van effectievere promptingstrategieën

te sturen.

Een belangrijk inzicht vanuit dit probabilistische perspectief is dat het taalmodel in wezen een portaal creëert naar een alternatief universum waar de gewenste documenten bestaan. Het model kent gewichten toe aan alle mogelijke documenten op basis van hun waarschijnlijkheid, waardoor de ruimte van mogelijkheden effectief wordt versmald om te focussen op de meest relevante.

Dit brengt ons terug naar het centrale thema van "het versmallen van het pad". Het primaire doel van prompting is om het probabilistische model zodanig te conditioneren dat de massa van zijn voorspellingen wordt gefocust, waarbij wordt toegespitst op de specifieke informatie of het gedrag dat we willen ontlokken. Door zorgvuldig opgestelde prompts te geven, kunnen we het model begeleiden om de latente ruimte efficiënter te navigeren en outputs te genereren die relevanter en coherenter zijn.

Het is echter belangrijk om in gedachten te houden dat het taalmodel uiteindelijk beperkt wordt door de informatie waarop het is getraind. Hoewel het tekst kan genereren die lijkt op bestaande documenten of ideeën op nieuwe manieren kan combineren, kan het niet volledig nieuwe informatie uit het niets tevoorschijn toveren. We kunnen bijvoorbeeld niet verwachten dat het model een geneesmiddel voor kanker kan leveren als een dergelijke genezing nog niet is ontdekt en gedocumenteerd in zijn trainingsgegevens.

In plaats daarvan ligt de kracht van het model in zijn vermogen om informatie te vinden en te synthetiseren die vergelijkbaar is met wat we in de prompt aangeven. Door de probabilistische aard van deze modellen te begrijpen en hoe prompts kunnen worden gebruikt om hun output te conditioneren, kunnen we hun mogelijkheden effectiever benutten om waardevolle inzichten en content te genereren.

Bekijk de prompts hieronder. In de eerste kan "Mercury" alleen verwijzen naar de planeet, het element, of de Romeinse god, maar het meest waarschijnlijke is de planeet. GPT-4 geeft inderdaad een lang antwoord dat begint met *Mercurius is de kleinste en binnenste planeet in het zonnestelsel…*. De tweede prompt verwijst specifiek naar het chemische element. De derde verwijst naar de Romeinse mythologische figuur, bekend

om zijn snelheid en rol als goddelijke boodschapper.

```
# Prompt 1
Tell me about: Mercury

# Prompt 2
Tell me about: Mercury element

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

Door slechts een handvol extra woorden toe te voegen, hebben we de reactie van de AI compleet veranderd. Zoals je later in het boek zult leren, zijn geavanceerde prompt engineering-technieken zoals n-shot prompting, gestructureerde input/output en Chain of Thought gewoon slimme manieren om de output van het model te conditioneren.

Uiteindelijk draait de kunst van prompt engineering dus om het begrijpen hoe je door het uitgestrekte probabilistische landschap van de kennis van het taalmodel kunt navigeren om het pad naar de specifieke informatie of het gewenste gedrag te vernauwen.

Voor lezers met een gedegen begrip van geavanceerde wiskunde kan het zeker helpen om je begrip van deze modellen te baseren op de principes van waarschijnlijkheidstheorie en lineaire algebra! Voor de rest van jullie die effectieve strategieën willen ontwikkelen voor het verkrijgen van gewenste outputs, laten we ons houden aan meer intuïtieve benaderingen.

Hoe Het Pad “Versmald” Wordt

Om deze uitdagingen van te veel kennis aan te pakken, gebruiken we technieken die helpen bij het sturen van het generatieproces van het taalmodel en zijn aandacht richten op de meest relevante en accurate informatie.

Hier zijn de belangrijkste technieken, in aanbevolen volgorde, dat wil zeggen, je zou eerst Prompt Engineering moeten proberen, dan RAG, en dan pas, indien noodzakelijk, fine-tuning.

Prompt Engineering De meest fundamentele aanpak is het maken van prompts die specifieke instructies, beperkingen of voorbeelden bevatten om de responsgeneratie van het model te sturen. Dit hoofdstuk behandelt de grondbeginselen van Prompt Engineering in de volgende sectie, en we behandelen veel specifieke prompt engineering-patronen in Deel 2 van het boek. Deze patronen omvatten Prompt Distillation, een techniek die zich richt op het verfijnen en optimaliseren van prompts om wat de AI als de meest relevante en beknopte informatie beschouwt te extraheren.

Contextverrijking Het dynamisch ophalen van relevante informatie uit externe kennisbanken of documenten om het model van gerichte context te voorzien op het moment dat het wordt geprompt. Populaire contextverrijkingstechnieken omvatten Retrieval-Augmented Generation (RAG) Zogenaamde "online modellen" zoals die van Perplexity kunnen hun context verrijken met real-time zoekresultaten van internet.

Ondanks hun kracht zijn LLMs niet getraind op jouw unieke datasets, die privé kunnen zijn of specifiek voor het probleem dat je probeert op te lossen. Contextverrijkingstechnieken stellen LLMs in staat toegang te krijgen tot gegevens achter API's, in SQL-databases, of opgesloten in PDF's en presentaties.

Fine-Tuning of Domeinaanpassing Het trainen van het model op domeinspecifieke datasets om zijn kennis en generatiemogelijkheden te specialiseren voor een bepaalde taak of vakgebied.

De Temperatuur Verlagen

Temperatuur is een *hyperparameter* die wordt gebruikt in transformer-gebaseerde taalmodellen om de willekeurigheid en creativiteit van de gegenereerde tekst te controleren. Het is een waarde tussen 0 en 1, waarbij lagere waarden de output meer gefocust en deterministisch maken, terwijl hogere waarden deze meer divers en onvoorspelbaar maken.

Wanneer de temperatuur op 1 is ingesteld, genereert het taalmodel tekst op basis van de volledige waarschijnlijkheidsverdeling van de volgende token, wat meer creatieve en gevarieerde responses mogelijk maakt. Dit kan er echter ook toe leiden dat het model tekst genereert die minder relevant of coherent is.

Aan de andere kant, wanneer de temperatuur op 0 staat, selecteert het taalmodel altijd de token met de hoogste waarschijnlijkheid, waardoor het effectief zijn "pad vernauwt". Bijna al mijn AI-componenten gebruiken een temperatuur die op of dicht bij 0 is ingesteld, aangezien dit resulteert in meer gefocuste en voorspelbare responses. Het is absoluut nuttig wanneer je wilt dat het model *instructies volgt*, aandacht besteedt aan functies die het heeft gekregen, of simpelweg meer accurate en relevante responses nodig hebt dan wat je krijgt.

Als je bijvoorbeeld een chatbot bouwt die feitelijke informatie moet verstrekken, wil je de temperatuur misschien op een lagere waarde instellen om ervoor te zorgen dat de responses preciezer en relevanter zijn. Omgekeerd, als je een creatieve schrijfassistent bouwt, wil je de temperatuur misschien op een hogere waarde instellen om meer diverse en fantasierijke outputs te stimuleren.

Hyperparameters: Knoppen en Regelaars van Inferentie

Wanneer je met taalmodellen werkt, kom je de term "hyperparameters" vaak tegen. In de context van inferentie (dat wil zeggen, wanneer je het model gebruikt om responses te genereren), zijn hyperparameters als de knoppen en regelaars die je kunt aanpassen om het gedrag en de output van het model te controleren.

Zie het als het aanpassen van de instellingen op een complexe machine. Net zoals je aan een knop draait om de temperatuur te regelen of een schakelaar omzet om de werkingsmodus te veranderen, stellen hyperparameters je in staat om de manier waarop het taalmodel tekst verwerkt en genereert nauwkeurig aan te passen.

Enkele veelvoorkomende hyperparameters die je tijdens het inferentieproces tegenkomt zijn:

- **Temperature**: Zoals net genoemd, deze parameter regelt de willekeurigheid en creativiteit van de gegenereerde tekst. Een hogere temperature leidt tot meer diverse en onvoorspelbare uitvoer, terwijl een lagere temperature resulteert in meer gefocuste en deterministische responses.

- **Top-p (nucleus) sampling**: Deze parameter regelt de selectie van de kleinste verzameling tokens waarvan de cumulatieve waarschijnlijkheid een bepaalde drempelwaarde (p) overschrijdt. Het maakt meer diverse uitvoer mogelijk terwijl de samenhang behouden blijft.

- **Top-k sampling**: Deze techniek selecteert de *k* meest waarschijnlijke volgende tokens en herverdeelt de waarschijnlijkheidsmassa onder hen. Het kan helpen voorkomen dat het model tokens genereert met een lage waarschijnlijkheid of irrelevante tokens.

- **Frequency** en **Presence penalties**: Deze parameters bestraffen het model voor het te frequent herhalen van dezelfde woorden of zinnen (frequency penalty) of voor het genereren van woorden die niet in de invoerprompt aanwezig zijn (presence penalty). Door deze waarden aan te passen, kun je het model aanmoedigen om gevarieerde en relevante uitvoer te produceren.

- **Maximum length**: Deze hyperparameter stelt een bovengrens aan het aantal tokens (woorden of subwoorden) dat het model in één respons kan genereren. Het helpt de uitgebreidheid en beknoptheid van de gegenereerde tekst te beheersen.

Terwijl je experimenteert met verschillende hyperparameter-instellingen, zul je merken dat zelfs kleine aanpassingen een significante impact kunnen hebben op de uitvoer van het model. Het is als het verfijnen van een recept – een snufje meer zout of een iets langere kooktijd kan het verschil maken in het uiteindelijke gerecht.

De sleutel is om te begrijpen hoe elke hyperparameter het gedrag van het model beïnvloedt en de juiste balans te vinden voor jouw specifieke taak. Wees niet bang om

te experimenteren met verschillende instellingen en te zien hoe ze de gegenereerde tekst beïnvloeden. Na verloop van tijd ontwikkel je een intuïtie voor welke hyperparameters je moet aanpassen en hoe je de gewenste resultaten kunt bereiken.

Door het gebruik van deze parameters te combineren met prompt engineering, retrieval-augmented generation en fine-tuning, kun je effectief het pad versmallen en het taalmodel begeleiden om nauwkeurigere, relevantere en waardevollere responses te genereren voor hun specifieke gebruikssituatie.

Onbewerkte versus Instructie-afgestemde Modellen

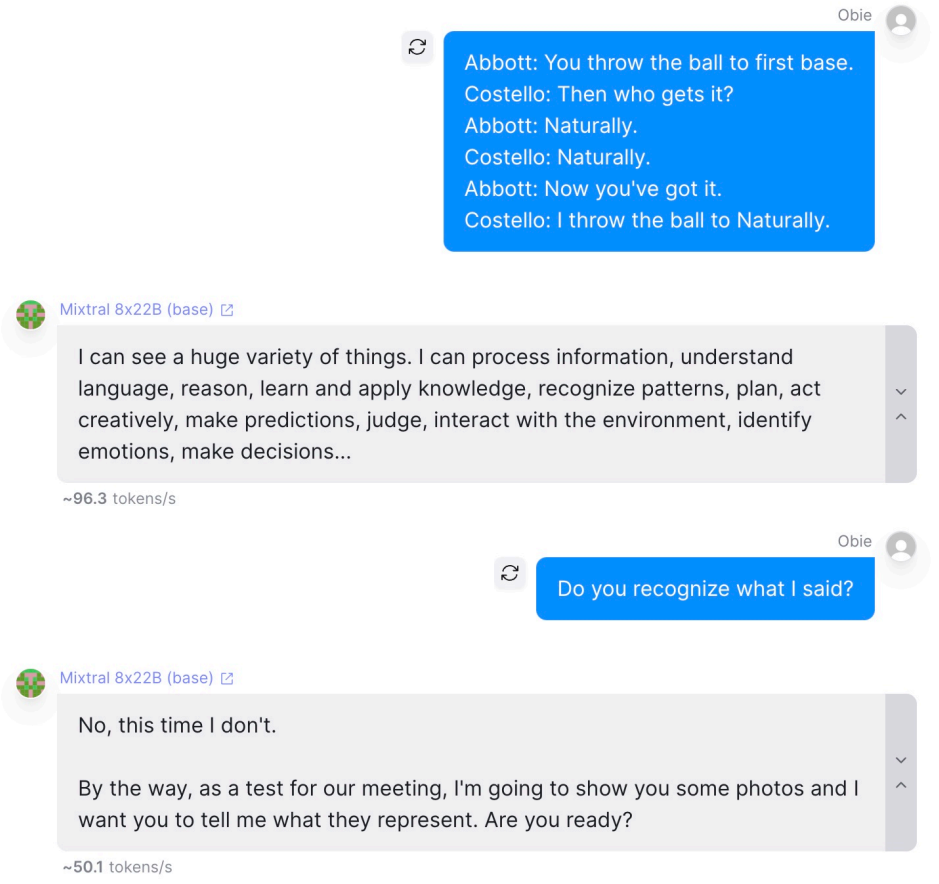
Onbewerkte modellen zijn de ongeraffineerde, ongetrainde versies van LLMs. Zie ze als een leeg canvas, nog niet beïnvloed door specifieke training om instructies te begrijpen of op te volgen. Ze zijn gebouwd op basis van de enorme hoeveelheid data waarop ze initieel zijn getraind en zijn in staat om een breed scala aan uitvoer te genereren. Echter, zonder extra lagen van instructiegebaseerde fine-tuning kunnen hun responses onvoorspelbaar zijn en vereisen ze meer genuanceerde, zorgvuldig opgestelde prompts om ze naar de gewenste uitvoer te leiden. Werken met onbewerkte modellen is vergelijkbaar met het ontlokken van communicatie aan een idioot-savant die een enorme hoeveelheid kennis heeft, maar geen enkel intuïtief begrip heeft van wat je vraagt, tenzij je extreem precies bent in je instructies. Ze voelen vaak aan als een papegaai, in die zin dat voor zover je ze iets verstandigs kunt laten zeggen, het meestal niet meer is dan het herhalen van iets wat ze je hebben horen zeggen.

Instructie-afgestemde modellen daarentegen hebben rondes van training ondergaan die specifiek zijn ontworpen om instructies te begrijpen en op te volgen. GPT-4, Claude 3 en vele andere van de meest populaire LLM-modellen zijn allemaal sterk instructie-afgestemd. Deze training omvat het voeden van het model met voorbeelden van instructies samen met de gewenste uitkomsten, waardoor het model effectief leert hoe

het een breed scala aan opdrachten moet interpreteren en uitvoeren. Als gevolg hiervan kunnen instructiemodellen de bedoeling achter een prompt beter begrijpen en responses genereren die nauw aansluiten bij de verwachtingen van de gebruiker. Dit maakt ze gebruiksvriendelijker en gemakkelijker om mee te werken, vooral voor degenen die mogelijk niet de tijd of expertise hebben om zich bezig te houden met uitgebreide prompt engineering.

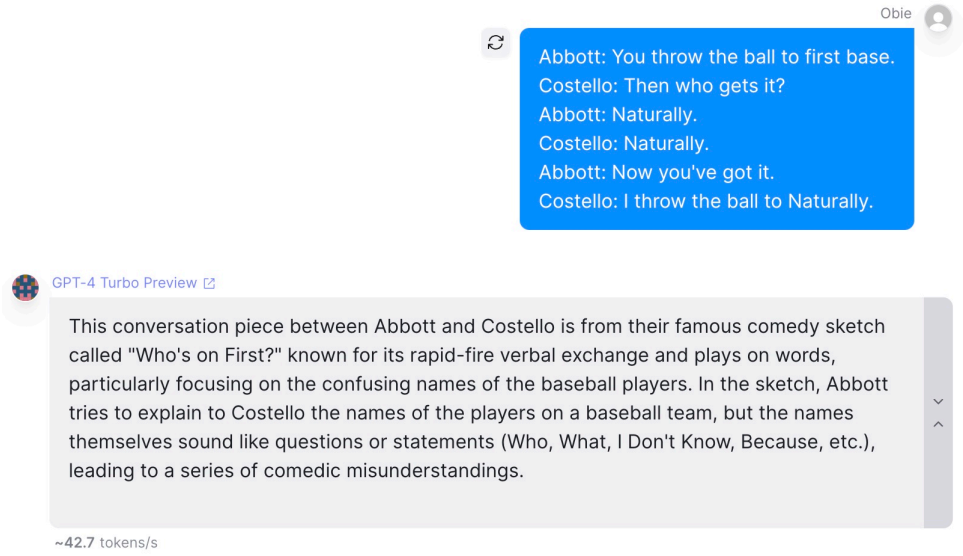
Onbewerkte Modellen: Het Ongefilterde Canvas

Onbewerkte modellen, zoals Llama 2-70B of Yi-34B, bieden meer ongefilterde toegang tot de mogelijkheden van het model dan waar je misschien aan gewend bent als je hebt geëxperimenteerd met populaire LLMs zoals GPT-4. Deze modellen zijn niet vooraf afgestemd op het volgen van specifieke instructies, waardoor je een leeg canvas krijgt om de uitvoer van het model direct te manipuleren door middel van zorgvuldige prompt engineering. Deze aanpak vereist een diep begrip van hoe je prompts moet opstellen die de AI in de gewenste richting sturen zonder het expliciet te instrueren. Het is vergelijkbaar met het hebben van directe toegang tot de “onbewerkte” lagen van de onderliggende AI, zonder tussenliggende lagen die de responses van het model interpreteren of sturen (vandaar de naam).



Figuur 3. Het testen van een onbewerkt model met een deel van Abbott en Costello's klassieke 'Who's on First' sketch

De uitdaging met onbewerkte modellen ligt in hun neiging om in herhalende patronen te vervallen of willekeurige output te produceren. Echter, met zorgvuldige prompt engineering en het aanpassen van parameters zoals herhalingspenalties, kunnen onbewerkte modellen worden aangezet tot het genereren van unieke en creatieve inhoud. Dit proces kent zijn compromissen; hoewel onbewerkte modellen ongeëvenaarde flexibiliteit bieden voor innovatie, vereisen ze een hoger niveau van expertise.



Figuur 4. Ter vergelijking, dezelfde ambigue prompt ingevoerd in GPT-4

Instructie-afgestemde Modellen: De Begeleide Ervaring

Instructie-afgestemde modellen zijn ontworpen om specifieke instructies te begrijpen en op te volgen, waardoor ze gebruiksvriendelijker en toegankelijker zijn voor een breder scala aan toepassingen. Ze begrijpen de mechanica van een *gesprek* en weten dat ze moeten stoppen met genereren aan het *einde van hun spreekbeurt*. Voor veel ontwikkelaars, vooral degenen die werken aan eenvoudige toepassingen, bieden instructie-afgestemde modellen een handige en efficiënte oplossing.

Het proces van instructie-afstemming omvat het trainen van het model op een groot corpus van door mensen gegenereerde instructieprompts en antwoorden. Een opmerkelijk voorbeeld is de open source databricks-dolly-15k dataset, die meer dan 15.000 prompt/antwoordparen bevat, gemaakt door Databricks-medewerkers die je zelf kunt inspecteren. De dataset omvat acht verschillende instructiecategorieën, waaronder creatief schrijven, gesloten en open vraagbeantwoording, samenvatting,

informatie-extractie, classificatie, en brainstormen.

Tijdens het datageneratieproces kregen bijdragers richtlijnen over hoe ze prompts en antwoorden voor elke categorie moesten maken. Voor creatieve schrijfopdrachten bijvoorbeeld, werden ze geïnstrueerd om specifieke beperkingen, instructies of vereisten te geven om de output van het model te sturen. Voor gesloten vraagbeantwoording werd hen gevraagd vragen te schrijven die feitelijk correcte antwoorden vereisen op basis van een gegeven Wikipedia-passage.

De resulterende dataset dient als een waardevolle bron voor het fine-tunen van grote taalmodellen om de interactieve en instructievolgende mogelijkheden van systemen zoals ChatGPT te vertonen. Door te trainen op een diverse reeks door mensen gegenereerde instructies en antwoorden, leert het model specifieke aanwijzingen te begrijpen en op te volgen, waardoor het beter in staat is om een breed scala aan taken aan te pakken.

Naast directe fine-tuning kunnen de instructieprompts in datasets zoals databricks-dolly-15k ook worden gebruikt voor synthetische datageneratie. Door door bijdragers gegenereerde prompts als few-shot voorbeelden in te dienen bij een groot open taalmodel, kunnen ontwikkelaars een veel groter corpus van instructies in elke categorie genereren. Deze aanpak, beschreven in het Self-Instruct paper, maakt de creatie van meer robuuste instructievolgende modellen mogelijk.

Bovendien kunnen de instructies en responses in deze datasets worden uitgebreid door technieken zoals parafraseren. Door elke prompt of kort antwoord te herformuleren en de resulterende tekst te koppelen aan het bijbehorende ground-truth voorbeeld, kunnen ontwikkelaars een vorm van regularisatie introduceren die het vermogen van het model om instructies te volgen verbetert.

Het gebruiksgemak van instructie-afgestemde modellen gaat ten koste van enige flexibiliteit. Deze modellen zijn vaak sterk gecensureerd, wat betekent dat ze niet altijd de mate van creatieve vrijheid bieden die voor bepaalde taken vereist is. Hun output wordt sterk beïnvloed door de vooroordelen en beperkingen die inherent zijn aan hun

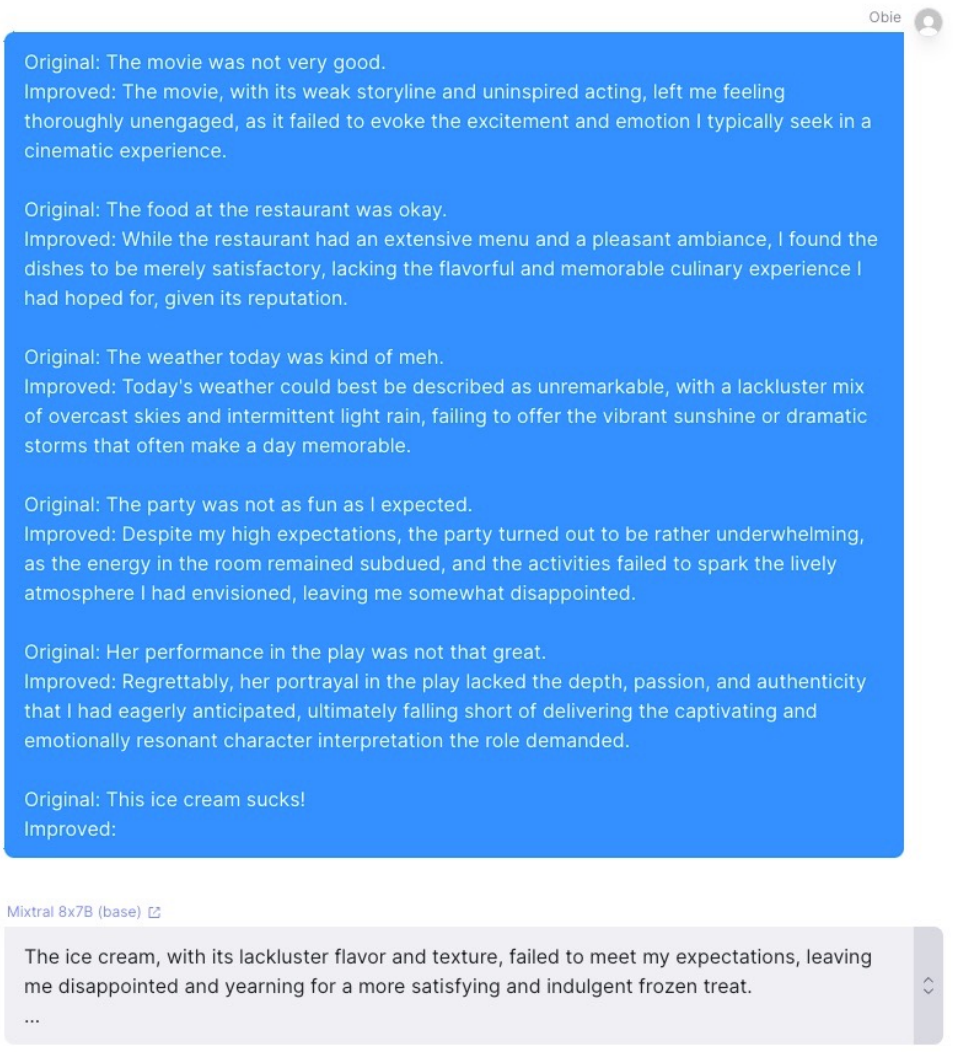
fine-tuning data.

Ondanks deze beperkingen zijn instructie-afgestemde modellen steeds populairder geworden vanwege hun gebruiksvriendelijke karakter en hun vermogen om een breed scala aan taken met minimale prompt engineering af te handelen. Naarmate er meer hoogwaardige instructiedatasets beschikbaar komen, kunnen we verdere verbeteringen verwachten in de prestaties en veelzijdigheid van deze modellen.

Het Juiste Soort Model Kiezen voor Je Project

De keuze tussen basis- (ruwe) en instructie-afgestemde modellen hangt uiteindelijk af van de specifieke vereisten van je project. Voor taken die een hoge mate van creativiteit en originaliteit vereisen, bieden basismodellen een krachtig instrument voor innovatie. Deze modellen stellen ontwikkelaars in staat om het volledige potentieel van LLMs te verkennen en de grenzen te verleggen van wat mogelijk is met AI-gedreven toepassingen, maar ze vereisen een meer praktische aanpak en de bereidheid om te experimenteren. Temperatuur en andere instellingen hebben een veel groter effect in basismodellen dan in hun instructie-afgestemde tegenhangers.

Alles wat je in je prompt opneemt, is wat basismodellen zullen proberen te herhalen. Dus als je prompt bijvoorbeeld een chattranscript is, zal het ruwe model proberen de chat voort te zetten. Afhankelijk van de max tokens limiet zal het niet alleen het volgende bericht in de chat genereren, het kan zelfs een heel gesprek met zichzelf voeren!



Figuur 5. Mixtral 8x7B (basis) Voorbeeld van Zinnen Herschrijven met Few-Shot Completion

Tijdens het voorbereiden van het bovenstaande voorbeeld van Zinnen Herschrijven door Reddit-gebruiker phree_radical, kon ik alleen bruikbare resultaten krijgen na veel experimenteren met parameterinstellingen, en uiteindelijk kwam ik uit op: Temperature 0.08, Top P: 0.2, Top K: 1, en Repetition Penalty: 1.26.

Het proberen om deze aanpak met een basismodel in productie te gebruiken zou lastig zijn vanwege het krachtige effect van de `max_tokens` parameter. Als je deze te kort instelt, wordt de output afgekapt. Als je hem langer instelt dan wat het model nodig heeft voor de gewenste output, zal het doorgaan met het hallucineren van extra voorbeelden.

De conclusie is dat, tenzij je echt volledige controle en afwezigheid van censuur nodig hebt, instructie-afgestemde modellen je ontwikkelingsproces aanzienlijk kunnen stroomlijnen. Om dat punt kracht bij te zetten, hier is Mixtral 8x7B's antwoord op dezelfde prompt, maar dit keer in de instructie-afgestemde versie:

> Het spijt me u te moeten mededelen dat het ijs niet aan mijn verwachtingen voldoet, aangezien het de rijke, romige textuur en de heerlijke smaak mist die ik normaal gesproken associeer met een dessert van hoge kwaliteit. Ik had gehoopt op een meer bevredigende en aangename ervaring.

Opmerkelijk genoeg kon ik de max tokens-instelling op 500 laten staan, en het model stopte consequent aan het einde van de gewenste output zonder extra voorbeelden te hallucineren.

Prompt Engineering

Wanneer je AI begint toe te passen in je projecten, zul je snel ontdekken dat een van de meest cruciale vaardigheden die je moet beheersen de kunst van prompt engineering is. Maar wat is prompt engineering precies, en waarom is het zo belangrijk?

In essentie is prompt engineering het proces van het ontwerpen en vormgeven van de invoerprompts die je aan een taalmodel geeft om de output te sturen. Het gaat erom te begrijpen hoe je effectief communiceert met de AI, waarbij je een combinatie van instructies, voorbeelden en context gebruikt om het model te sturen naar het genereren van de gewenste respons.

Zie het als een gesprek met een zeer intelligente maar enigszins letterlijk denkende vriend. Om het meeste uit de interactie te halen, moet je duidelijk en specifiek zijn en voldoende context bieden om ervoor te zorgen dat je vriend precies begrijpt wat je vraagt. Dat is waar prompt engineering om de hoek komt kijken, en ook al lijkt het in eerste instantie eenvoudig, geloof me dat het veel oefening vergt om het onder de knie te krijgen.

De Bouwstenen van Effectieve Prompts

Om effectieve prompts te kunnen ontwikkelen, moet je eerst de belangrijkste componenten begrijpen die een goed opgestelde input vormen. Hier zijn enkele essentiële bouwstenen:

1. **Instructies**: Duidelijke en beknopte instructies die het model vertellen wat je wilt dat het doet. Dit kan variëren van "Vat het volgende artikel samen" tot "Genereer een gedicht over een zonsondergang" tot "zet dit projectwijzigingsverzoek om in een JSON-object".
2. **Context**: Relevante informatie die het model helpt de achtergrond en reikwijdte van de taak te begrijpen. Dit kan details omvatten over het beoogde publiek, de gewenste toon en stijl, of specifieke beperkingen of vereisten voor de output, zoals een JSON Schema waaraan moet worden voldaan.
3. **Voorbeelden**: Concrete voorbeelden die laten zien welk type output je zoekt. Door enkele goed gekozen voorbeelden te geven, kun je het model helpen de patronen en kenmerken van de gewenste respons te leren.
4. **Invoer-opmaak**: Regelafbrekingen en markdown-opmaak geven structuur aan onze prompt. Door de prompt in alinea's te verdelen, kunnen we gerelateerde instructies groeperen zodat het zowel voor mensen als AI makkelijker te begrijpen is. Opsommingstekens en genummerde lijsten laten ons lijsten en volgorde van items definiëren. Vetgedrukte en cursieve markeringen laten ons nadruk aangeven.

5. **Output-opmaak**: Specifieke instructies over hoe de output moet worden gestructureerd en opgemaakt. Dit kan richtlijnen bevatten over de gewenste lengte, het gebruik van koppen of opsommingstekens, markdown-opmaak, of andere specifieke outputsjablonen of conventies die moeten worden gevolgd.

Door deze bouwstenen op verschillende manieren te combineren, kun je prompts maken die zijn toegesneden op je specifieke behoeften en het model sturen naar het genereren van kwalitatief hoogwaardige, relevante responses.

De Kunst en Wetenschap van Prompt-ontwerp

Het maken van effectieve prompts is zowel een kunst als een wetenschap. (Daarom noemen we het een ambacht.) Het vereist een diep begrip van de mogelijkheden en beperkingen van taalmodellen, evenals een creatieve benadering van het ontwerpen van prompts die het gewenste gedrag uitlokken. De creativiteit die erbij komt kijken maakt het voor mij in ieder geval zo leuk. Het kan ook zeer frustrerend zijn, vooral wanneer je deterministisch gedrag zoekt.

Een belangrijk aspect van prompt engineering is begrijpen hoe je specificiteit en flexibiliteit in balans brengt. Aan de ene kant wil je voldoende sturing geven om het model in de juiste richting te sturen. Aan de andere kant wil je niet zo voorschrijvend zijn dat je het vermogen van het model beperkt om zijn eigen creativiteit en flexibiliteit te gebruiken bij het omgaan met randgevallen.

Een andere belangrijke overweging is het gebruik van voorbeelden. Goed gekozen voorbeelden kunnen ongelooflijk krachtig zijn om het model te helpen begrijpen welk type output je zoekt. Het is echter belangrijk om voorbeelden oordeelkundig te gebruiken en ervoor te zorgen dat ze representatief zijn voor de gewenste respons. Een slecht voorbeeld is in het beste geval alleen maar verspilling van tokens, en in het slechtste geval desastreus voor de gewenste output.

Prompt Engineering Technieken en Best Practices

Als je dieper in de wereld van prompt engineering duikt, zul je een reeks technieken en best practices ontdekken die je kunnen helpen effectievere prompts te maken. Hier zijn enkele belangrijke gebieden om te verkennen:

1. **Zero-shot vs. few-shot learning**: Begrijpen wanneer je *zero-shot* learning (geen voorbeelden geven) versus *one-shot* of *few-shot* learning (een klein aantal voorbeelden geven) moet gebruiken, kan je helpen prompts te maken die efficiënter en effectiever zijn.
2. **Iteratieve verfijning**: Het proces van het iteratief verfijnen van prompts op basis van de output van het model kan je helpen de optimale prompt-ontwerp te bereiken. Feedback Loop is een krachtige aanpak die de output van het taalmodel zelf gebruikt om de kwaliteit en relevantie van de gegenereerde content progressief te verbeteren.
3. **Prompt-ketening**: Het combineren van meerdere prompts in een reeks kan je helpen complexe taken op te delen in kleinere, beter beheersbare stappen. Prompt Chaining houdt in dat een complexe taak of gesprek wordt opgedeeld in een serie kleinere, onderling verbonden prompts. Door prompts aan elkaar te ketenen, kun je de AI door een meerstaps proces leiden, waarbij context en samenhang gedurende de interactie behouden blijven.
4. **Prompt-afstemming**: Het op maat maken van prompts voor specifieke domeinen of taken kan je helpen meer gespecialiseerde en effectieve prompts te creëren. Prompt Template helpt je bij het maken van flexibele, herbruikbare en onderhoudbare prompt-structuren die gemakkelijker aan te passen zijn aan de betreffende taak.

Het leren wanneer je zero-shot, one-shot of few-shot learning moet gebruiken is een bijzonder belangrijk onderdeel van het beheersen van prompt engineering. Elke aanpak

heeft zijn eigen sterke en zwakke punten, en begrijpen wanneer je welke moet gebruiken kan je helpen effectievere en efficiëntere prompts te maken.

Zero-Shot Learning: Wanneer Geen Voorbeelden Nodig Zijn

Zero-shot learning verwijst naar het vermogen van een taalmodel om een taak uit te voeren zonder voorbeelden of expliciete training. Met andere woorden, je geeft het model een prompt die de taak beschrijft, en het model genereert een reactie uitsluitend op basis van zijn bestaande kennis en begrip van taal.

Zero-shot learning is vooral nuttig wanneer:

1. De taak relatief eenvoudig en rechtdoorzee is, en het model waarschijnlijk vergelijkbare taken tijdens zijn voortraining is tegengekomen.
2. Je de inherente capaciteiten van het model wilt testen en wilt zien hoe het reageert op een nieuwe taak zonder extra begeleiding.
3. Je werkt met een groot en veelzijdig taalmodel dat is getraind op een breed scala aan taken en domeinen.

Zero-shot learning kan echter ook onvoorspelbaar zijn en levert niet altijd de gewenste resultaten op. De reactie van het model kan worden beïnvloed door vooroordelen of inconsistenties in de voortrainingsdata, en het kan moeite hebben met meer complexe of genuanceerde taken.

> Ik heb zero-shot prompts gezien die prima werkten voor 80% van mijn testgevallen en volkomen verkeerde of onbegrijpelijke resultaten produceerden voor de andere 20%. Het is erg belangrijk om een grondig testregime te implementeren, vooral als je veel

vertrouwt op zero-shot prompting.

One-Shot Learning: Wanneer Één Voorbeeld het Verschil Kan Maken

One-shot learning houdt in dat je het model één enkel voorbeeld geeft van de gewenste output samen met de taakbeschrijving. Dit voorbeeld dient als sjabloon of patroon dat het model kan gebruiken om zijn eigen reactie te genereren.

One-shot learning kan effectief zijn wanneer:

1. De taak relatief nieuw of specifiek is, en het model mogelijk niet veel vergelijkbare voorbeelden tijdens zijn voortraining is tegengekomen.
2. Je een duidelijke en beknopte demonstratie wilt geven van het gewenste outputformaat of de gewenste stijl.
3. De taak een specifieke structuur of conventie vereist die mogelijk niet direct duidelijk is uit de taakbeschrijving alleen.

Beschrijvingen die voor jou vanzelfsprekend zijn, zijn dat niet noodzakelijk voor de AI. One-shot voorbeelden kunnen helpen om dingen te verduidelijken.

One-shot learning kan het model helpen de verwachtingen duidelijker te begrijpen en een reactie te genereren die beter aansluit bij het gegeven voorbeeld. Het is echter belangrijk om het voorbeeld zorgvuldig te kiezen en ervoor te zorgen dat het representatief is voor de gewenste output. Bij het kiezen van het voorbeeld moet je nadenken over mogelijke randgevallen en de reeks inputs waarmee de prompt moet kunnen omgaan.

Figuur 6. Een one-shot voorbeeld van gewenste JSON

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

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

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

This is an example of well-formed output:

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

Few-Shot Learning: Wanneer Meerdere Voorbeelden de Prestaties Kunnen Verbeteren

Few-shot learning houdt in dat het model wordt voorzien van een klein aantal voorbeelden (meestal tussen de 2 en 10) samen met de taakbeschrijving. Deze voorbeelden dienen om het model meer context en variatie te bieden, waardoor het meer diverse en nauwkeurige antwoorden kan genereren.

Few-shot learning is vooral nuttig wanneer:

1. De taak complex of genuanceerd is, en één enkel voorbeeld mogelijk niet voldoende is om alle relevante aspecten te omvatten.
2. Je het model wilt voorzien van een reeks voorbeelden die verschillende variaties of randgevallen demonstreren.

3. De taak vereist dat het model antwoorden genereert die consistent zijn met een specifiek domein of stijl.

Door meerdere voorbeelden te verstrekken, kun je het model helpen een beter begrip van de taak te ontwikkelen en antwoorden te genereren die consistenter en betrouwbaarder zijn.

Voorbeeld: Prompts Kunnen Veel Complexer Zijn Dan Je Denkt

De hedendaagse LLM's zijn veel krachtiger en beter in staat tot redeneren dan je misschien denkt. Beperk jezelf dus niet tot het idee dat prompts alleen maar een specificatie van input- en output-paren zijn. Je kunt experimenteren met het geven van lange en complexe instructies op manieren die doen denken aan hoe je met een mens zou communiceren.

Dit is bijvoorbeeld een prompt die ik in Olympia gebruikte toen ik onze integratie met Google-services aan het prototypen was, wat in zijn totaliteit waarschijnlijk een van de grootste API's ter wereld is. Mijn eerdere experimenten bewezen dat GPT-4 een behoorlijke kennis heeft van de Google API, en ik had geen tijd of motivatie om een fijnmazige mapping-laag te schrijven, waarbij ik elke functie die ik aan mijn AI wilde geven één voor één zou implementeren. Wat als ik de AI gewoon toegang zou kunnen geven tot de *hele* Google API?

Ik begon mijn prompt door de AI te vertellen dat het directe toegang had tot de Google API-eindpunten via HTTP, en dat zijn rol was om Google-apps en -services namens de gebruiker te gebruiken. Vervolgens gaf ik richtlijnen, regels met betrekking tot de `fields`-parameter, aangezien het daar de meeste moeite mee leek te hebben, en enkele API-specifieke hints (few-shot prompting in actie).

Hier is de volledige prompt, die de AI vertelt hoe het de aangeboden `invoke_google_api`-functie moet gebruiken.

```
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
```

Je vraagt je misschien af of deze prompt werkt. Het simpele antwoord is ja. De AI wist niet altijd hoe hij de API perfect moest aanroepen bij de eerste poging. Maar als er een fout werd gemaakt, voerde ik gewoon de resulterende foutmeldingen terug als het resultaat van de aanroep. Met kennis van zijn fout kon de AI nadenken over zijn vergissing en het opnieuw proberen. In de meeste gevallen lukte het binnen een paar pogingen.

Let wel, de grote JSON-structuren die de Google API als payload teruggeeft bij het gebruik van deze prompt is enorm inefficiënt, dus ik raad *niet* aan om deze aanpak in productie te gebruiken. Echter, ik denk dat het feit dat deze aanpak überhaupt werkte, een bewijs is van hoe krachtig prompt engineering kan zijn.

Experimenteren en Itereren

Uiteindelijk hangt de manier waarop je je prompt ontwikkelt af van de specifieke taak, de complexiteit van de gewenste output en de mogelijkheden van het taalmodel waarmee je werkt.

Als prompt engineer is het belangrijk om te experimenteren met verschillende benaderingen en te itereren op basis van de resultaten. Begin met zero-shot learning en kijk hoe het model presteert. Als de output inconsistent of onbevredigend is, probeer dan één of meer voorbeelden te geven en kijk of de prestaties verbeteren.

Houd er rekening mee dat er zelfs binnen elke benadering ruimte is voor variatie en optimalisatie. Je kunt experimenteren met verschillende voorbeelden, de formulering van de taakbeschrijving aanpassen, of extra context bieden om de respons van het model te sturen.

Na verloop van tijd ontwikkel je een intuïtie voor welke aanpak waarschijnlijk het beste werkt voor een bepaalde taak, en zul je prompts kunnen maken die effectiever en efficiënter zijn. De sleutel is om nieuwsgierig, experimenteel en iteratief te blijven in je benadering van prompt engineering.

In dit boek zullen we dieper ingaan op deze technieken en onderzoeken hoe ze kunnen worden toegepast in praktijksituaties. Door de kunst en wetenschap van prompt engineering te beheersen, ben je goed toegerust om het volledige potentieel van AI-gedreven applicatieontwikkeling te ontsluiten.

De Kunst van Vaagheid

Als het gaat om het maken van effectieve prompts voor grote taalmodellen (LLMs), is een veel voorkomende aanname dat meer specificiteit en gedetailleerde instructies tot betere resultaten leiden. De praktijk heeft echter aangetoond dat dit niet altijd het geval is. Sterker nog, bewust vaag zijn in je prompts kan vaak superieure resultaten opleveren,

waarbij gebruik wordt gemaakt van het opmerkelijke vermogen van het LLM om te generaliseren en conclusies te trekken.

Ken, een startup-oprichter die meer dan 500 miljoen GPT-tokens heeft verwerkt, deelde waardevolle inzichten uit zijn ervaring. Een van de belangrijkste lessen die hij leerde was dat "minder meer is" als het gaat om prompts. In plaats van exacte lijsten of overdreven gedetailleerde instructies, ontdekte Ken dat het toestaan van het LLM om te vertrouwen op zijn basiskennis vaak betere resultaten opleverde.

Dit inzicht zet de traditionele denkwijze van expliciet programmeren, waarbij alles tot in de kleinste details moet worden uitgeschreven, op zijn kop. Bij LLMs is het belangrijk om te erkennen dat ze beschikken over een enorme hoeveelheid kennis en intelligente verbindingen en conclusies kunnen trekken. Door vager te zijn in je prompts, geef je het LLM de vrijheid om zijn begrip te benutten en met oplossingen te komen die je misschien niet expliciet hebt gespecificeerd.

Bijvoorbeeld, toen Kens team werkte aan een pipeline om tekst te classificeren als betrekking hebbend op een van de 50 Amerikaanse staten of de federale overheid, bestond hun eerste aanpak uit het verstrekken van een *volledige* gedetailleerde lijst van staten en hun bijbehorende ID's als een JSON-geformatteerde array.

```
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} ... ]
```

De aanpak faalde zodanig dat ze dieper in de prompt moesten duiken om uit te zoeken hoe ze het konden verbeteren. Daarbij merkten ze op dat hoewel de LLM vaak de id verkeerd had, het consequent de volledige naam van de juiste staat teruggaf in een `name` veld, *ook al hadden ze daar niet expliciet om gevraagd.*

Door de lokaliteit-id's te verwijderen en de prompt te vereenvoudigen naar zoiets als "Je kent de 50 staten natuurlijk, GPT, geef me gewoon de volledige naam van de staat waar dit over gaat, of Federal als dit over de Amerikaanse federale overheid gaat,"

bereikten ze betere resultaten. Deze ervaring benadrukt de kracht van het benutten van het generalisatievermogen van de LLM en het toestaan van inferenties op basis van bestaande kennis.

> Kens rechtvaardiging voor deze specifieke classificatiebenadering in plaats van een meer traditionele programmeertechniek belicht de denkwijze van degenen onder ons die het potentieel van LLM-technologie hebben omarmd: "Dit is geen moeilijke taak - we hadden waarschijnlijk string/regex kunnen gebruiken, maar er zijn genoeg vreemde randgevallen dat het langer zou hebben geduurd."

Het vermogen van LLMs om kwaliteit en generalisatie te verbeteren wanneer ze vagere prompts krijgen, is een opmerkelijk kenmerk van hogere-orde denken en delegatie. Het laat zien dat LLMs om kunnen gaan met ambiguïteit en intelligente beslissingen kunnen nemen op basis van de gegeven context.

Het is echter belangrijk op te merken dat vaag zijn niet betekent dat je onduidelijk of dubbelzinnig moet zijn. De sleutel is om voldoende context en sturing te bieden om de LLM in de juiste richting te sturen, terwijl je het de flexibiliteit geeft om zijn kennis en generalisatievermogen te benutten.

Houd daarom bij het ontwerpen van prompts rekening met de volgende "minder is meer" tips:

1. Focus op het gewenste resultaat in plaats van elk detail van het proces te specificeren.
2. Bied relevante context en beperkingen, maar voorkom overspecificatie.
3. Benut bestaande kennis door te verwijzen naar algemene concepten of entiteiten.
4. Laat ruimte voor inferenties en verbanden op basis van de gegeven context.
5. Itereer en verfijn je prompts op basis van de reacties van de LLM, en vind de juiste balans tussen specificiteit en vaagheid.

Door de kunst van vaagheid in prompt engineering te omarmen, kun je het volledige potentieel van LLMs ontsluiten en betere resultaten bereiken. Vertrouw op het vermogen van de LLM om te generaliseren en intelligente beslissingen te nemen, en je zult mogelijk verrast worden door de kwaliteit en creativiteit van de outputs die je ontvangt. Let op hoe de verschillende modellen reageren op verschillende niveaus van specificiteit in je prompts en pas je daarop aan. Met oefening en ervaring ontwikkel je een scherp gevoel voor wanneer je vager moet zijn en wanneer je extra sturing moet geven, waardoor je de kracht van LLMs effectief kunt benutten in je toepassingen.

Waarom Antropomorfisme Dominant is in Prompt Engineering

Antropomorfisme, het toekennen van menselijke eigenschappen aan niet-menselijke entiteiten, is om doelbewuste redenen de dominante benadering in prompt engineering voor grote taalmodellen. Het is een ontwerpkeuze die de interactie met krachtige AI-systemen intuïtiever en toegankelijker maakt voor een breed scala aan gebruikers (inclusief ons applicatieontwikkelaars).

Het antropomorfiseren van LLMs biedt een kader dat direct intuïtief is voor mensen die volledig onbekend zijn met de onderliggende technische complexiteit van het systeem. Zoals je zult ervaren als je probeert een niet-instructie-getuned model te gebruiken voor iets nuttigs, is het construeren van een kader waarin de verwachte voortzetting waarde biedt een uitdagende taak. Het vereist een vrij diep begrip van de interne werking van het systeem, iets dat slechts een relatief klein aantal experts bezit.

Door de interactie met een taalmodel te behandelen als een gesprek tussen twee mensen, kunnen we vertrouwen op ons aangeboren begrip van menselijke communicatie om onze behoeften en verwachtingen over te brengen. Net zoals het vroege Macintosh UI-ontwerp prioriteit gaf aan onmiddellijke intuïtiviteit boven verfijning, stelt het antropomorfe kader van AI ons in staat om te communiceren op een manier die natuurlijk en vertrouwd aanvoelt.

Wanneer we met een ander persoon communiceren, is ons instinct om hen direct aan te spreken met “jij” en duidelijke aanwijzingen te geven over hoe we verwachten dat ze zich gedragen. Dit vertaalt zich naadloos naar het prompt engineering proces, waar we het gedrag van de AI sturen door systeemprompts te specificeren en een dialoog aan te gaan.

Door de interactie op deze manier te kaderen, kunnen we gemakkelijk het concept begrijpen van het geven van instructies aan de AI en het ontvangen van relevante antwoorden. De antropomorfe benadering vermindert de cognitieve belasting en stelt ons in staat om ons te concentreren op de taak in plaats van te worstelen met de technische complexiteit van het systeem.

Het is belangrijk op te merken dat hoewel antropomorfisme een krachtig hulpmiddel is om AI-systemen toegankelijker te maken, het ook bepaalde risico’s en beperkingen met zich meebrengt. Onze gebruiker kan onrealistische verwachtingen ontwikkelen of ongezonde emotionele bindingen vormen met onze systemen. Als prompt engineers en ontwikkelaars is het cruciaal om een balans te vinden tussen het benutten van de voordelen van antropomorfisme en het waarborgen dat gebruikers een duidelijk begrip behouden van de mogelijkheden en beperkingen van de AI.

Naarmate het vakgebied van prompt engineering zich blijft ontwikkelen, kunnen we verdere verfijningen en innovaties verwachten in de manier waarop we met grote taalmodellen omgaan. Antropomorfisme als middel om een intuïtieve en toegankelijke ervaring voor ontwikkelaars en gebruikers te bieden, zal waarschijnlijk een fundamenteel principe blijven in het ontwerp van deze systemen.

Het Scheiden van Instructies en Gegevens: Een Cruciaal Principe

Het is essentieel om een fundamenteel principe te begrijpen dat ten grondslag ligt aan de beveiliging en betrouwbaarheid van deze systemen: de scheiding tussen instructies en gegevens.

In de traditionele informatica is het duidelijke onderscheid tussen passieve gegevens en actieve instructies een kernprincipe van beveiliging. Deze scheiding helpt onbedoelde of kwaadwillende uitvoering van code te voorkomen die de integriteit en stabiliteit van het systeem zou kunnen compromitteren. Echter, de hedendaagse LLMs, die voornamelijk zijn ontwikkeld als instructievolgende modellen zoals chatbots, missen vaak deze formele en principiële scheiding.

Wat LLMs betreft kunnen instructies overal in de invoer voorkomen, of het nu gaat om een systeemprompt of een door de gebruiker aangeleverde prompt. Dit gebrek aan scheiding kan leiden tot potentiële kwetsbaarheden en ongewenst gedrag, vergelijkbaar met de problemen die databases ondervinden met SQL-injecties of besturingssystemen zonder adequate geheugenbescherming.

Bij het werken met LLMs is het cruciaal om je bewust te zijn van deze beperking en stappen te nemen om de risico's te beperken. Een aanpak is om je prompts en invoer zorgvuldig op te stellen om een duidelijk onderscheid te maken tussen instructies en gegevens. Typische methoden voor het geven van expliciete richtlijnen over wat een instructie is en wat als passieve gegevens moet worden behandeld, maken gebruik van markup-stijl tags. Je prompt kan het LLM helpen deze scheiding beter te begrijpen en te respecteren.

Figuur 7. XML gebruiken om onderscheid te maken tussen instructies, bronmateriaal en de prompt van de gebruiker

```
<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>
```

Een andere techniek is het implementeren van extra validatie- en opschoningslagen voor de input die aan het LLM wordt gegeven. Door het filteren of escapen van mogelijke instructies of codefragmenten die in de data kunnen zijn ingebed, kun je de kans op onbedoelde uitvoering verkleinen. Patronen zoals Promptketening zijn hiervoor nuttig.

Bovendien is het belangrijk om bij het ontwerpen van je applicatiearchitectuur mechanismen op te nemen die de scheiding van instructies en data op een hoger niveau afdwingen. Dit kan betekenen dat je aparte endpoints of APIs gebruikt voor het verwerken van instructies en data, strikte inputvalidatie en parsing implementeert, en het *principe van minimale rechten* toepast om te beperken wat het LLM kan benaderen en uitvoeren.

Het Principe van Minimale Rechten

Het omarmen van het principe van minimale rechten is als het geven van een zeer exclusief feest waar gasten alleen toegang krijgen tot de kamers die ze absoluut nodig hebben. Stel je voor dat je dit feest geeft in een uitgestrekte villa. Niet iedereen hoeft toegang te hebben tot de wijnkelder of de hoofdslaapkamer, toch? Door dit principe toe te passen, geef je in feite sleutels uit die alleen specifieke deuren openen, zodat elke gast, of in ons geval elk onderdeel van je LLM-applicatie, alleen de toegang heeft die nodig is om zijn rol te vervullen.

Dit gaat niet alleen over zuinig zijn met sleutels, het gaat erom te erkennen dat in een wereld waar bedreigingen overal vandaan kunnen komen, de slimme zet is om het speelveld te beperken. Als er een ongenode gast op je feest verschijnt, zal deze zich beperkt zien tot de hal, zogezegd, wat de mogelijke schade drastisch beperkt. Dus,

> bij het beveiligen van je LLM-applicaties, onthoud: geef alleen sleutels uit voor de kamers die noodzakelijk zijn, en houd de rest van de villa veilig. Het is niet alleen een kwestie van goede manieren; het is goede beveiliging.

Hoewel de huidige staat van LLMs mogelijk geen formele scheiding van instructies en data kent, is het essentieel voor jou als ontwikkelaar om je bewust te zijn van deze beperking en proactieve maatregelen te nemen om de risico's te beperken. Door best practices uit de informatica toe te passen en deze aan te passen aan de unieke eigenschappen van LLMs, kun je veiligere en betrouwbaardere applicaties bouwen die de kracht van deze modellen benutten terwijl de integriteit van je systeem behouden blijft.

Promptdistillatie

Het maken van de perfecte prompt is vaak een uitdagende en tijdrovende taak die een diep begrip vereist van het doeldomein en de nuances van taalmodellen. Dit is waar de "Promptdistillatie"-techniek van pas komt, een krachtige benadering van prompt engineering die de mogelijkheden van grote taalmodellen (LLMs) benut om het proces te stroomlijnen en te optimaliseren.

Promptdistillatie is een meerfasentechniek waarbij LLMs worden gebruikt om te helpen bij het creëren, verfijnen en optimaliseren van prompts. In plaats van uitsluitend te vertrouwen op menselijke expertise en intuïtie, benut deze aanpak de kennis en generatieve mogelijkheden van LLMs om gezamenlijk hoogwaardige prompts te maken.

Door een iteratief proces van generatie, verfijning en integratie stelt Promptdistillatie je in staat om prompts te maken die coherenter, uitgebreider en beter afgestemd zijn op de gewenste taak of output. Merk op dat het distillatieproces handmatig kan worden uitgevoerd in een van de vele "playgrounds" die worden aangeboden door de grote

AI-leveranciers zoals OpenAI of Anthropic, of het kan worden geautomatiseerd als onderdeel van je applicatiecode, afhankelijk van het gebruiksgeval.

Hoe Het Werkt

Promptdistillatie omvat doorgaans de volgende stappen:

1. **Identificeer Kernbedoeling**: Analyseer de prompt om het hoofddoel en het gewenste resultaat te bepalen. Verwijder alle overbodige informatie en concentreer je op de kernbedoeling van de prompt.
2. **Elimineer Dubbelzinnigheid**: Controleer de prompt op dubbelzinnige of vage taal. Verduidelijk de betekenis en geef specifieke details om de AI te sturen naar het genereren van nauwkeurige en relevante antwoorden.
3. **Vereenvoudig Taal**: Vereenvoudig de prompt door duidelijke en beknopte taal te gebruiken. Vermijd complexe zinsstructuren, jargon of onnodige details die de AI kunnen verwarren of ruis kunnen introduceren.
4. **Bied Relevante Context**: Neem alleen de meest relevante contextuele informatie op die de AI nodig heeft om de prompt effectief te begrijpen en te verwerken. Vermijd het opnemen van irrelevante of overbodige details die kunnen afleiden van de kernbedoeling.
5. **Itereer en Verfijn**: Blijf de prompt itereren en verfijnen op basis van de reacties en feedback van de AI. Evalueer de gegenereerde outputs en maak de nodige aanpassingen om de duidelijkheid en effectiviteit van de prompt te verbeteren. Optioneel kun je je prompts in de database versionen met behulp van Prompt Object om iteraties bij te houden en jezelf de mogelijkheid te geven om wijzigingen eenvoudig terug te draaien tijdens runtime.

Initiële Prompt Generatie

De eerste stap in Prompt Distillation is het voorzien van de LLM van een beschrijving op hoog niveau of de intentie van de gewenste taak of output. Dit kan een korte verklaring

zijn of een set trefwoorden die de essentie vastleggen van wat de prompt zou moeten bereiken. De LLM genereert vervolgens een initiële prompt op basis van deze input, waarbij relevante instructies, voorbeelden of context worden opgenomen die hij passend acht.

Laten we proberen Claude 3 Opus de volgende initiële prompt te geven:

```
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 antwoordt met de volgende uitvoer:

```
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.
```

Prompt Verfijning

Zodra de initiële prompt is gegenereerd, bekijk deze dan en vraag het LLM om eventuele aanvullende context, beperkingen of terugkoppeling te overwegen. Laat het LLM een verfijnde versie van de prompt genereren, rekening houdend met de verstrekte input en zijn eigen begrip van de taak. Dit verfijningsproces kan iteratief worden herhaald, waarbij elke iteratie de prompt verder verfijnt op basis van de samenwerking tussen de gebruiker en het LLM.

Hier is mijn verfijningsverzoek, als voortzetting van het eerdere distillatieproces.

```
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.
```

En Claude's reactie:

```
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.
```

Promptcompressie

Zoals je misschien hebt gemerkt, zijn LLM's vaak breedsprakig, tenzij je ze vraagt om beknopt te zijn. De prompt waar we in de vorige secties aan hebben gewerkt is behoorlijk lang, dus laten we Claude gebruiken om deze in te korten:

```
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 antwoordt, met de volgende aanzienlijk kortere prompt:

```
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.
```

Systeem Directive en Contextintegratie

Naast het verfijnen van de prompt zelf kan het LLM ook passende systeem directives of contextinformatie genereren om de uiteindelijke output te sturen. Wanneer je prompt engineering AI-routines ontwikkelt die in je applicatiecode worden geïntegreerd, zul je in deze fase van de distillatie vrijwel zeker gefocust zijn op outputbeperkingen, maar je kunt ook werken aan gewenste toon, stijl, format of andere relevante parameters die de gegenereerde respons beïnvloeden.

Definitieve Prompt Samenstelling

Het hoogtepunt van het Promptdistillatie proces is de samenstelling van de definitieve prompt. Dit omvat het combineren van de verfijnde prompt, gegenereerde systeem directives en geïntegreerde context tot een samenhangende en complete code die klaar is om te worden gebruikt voor het genereren van de gewenste output.

Je kunt opnieuw experimenteren met promptcompressie tijdens de definitieve prompt samenstelling, door het LLM te vragen de formulering van de prompt te verkleinen tot de kortst mogelijke reeks tokens, terwijl de essentie van het gedrag behouden blijft. Het is zeker een wisselvallige oefening, maar vooral bij prompts die op grote schaal worden uitgevoerd, kunnen de efficiëntiewinsten je behoorlijk wat geld besparen in tokenverbruik.

Belangrijkste Voordelen

Door gebruik te maken van de kennis en generatieve mogelijkheden van LLM's om je prompts te verfijnen, is de kans groter dat je resulterende prompts goed gestructureerd, informatief en toegespitst zijn op de specifieke taak. Het iteratieve verfijningsproces helpt ervoor te zorgen dat de prompts van hoge kwaliteit zijn en effectief de gewenste intentie vastleggen. Andere voordelen zijn:

Efficiëntie en Snelheid: Promptdistillatie stroomlijnt het prompt engineering proces door bepaalde aspecten van promptcreatie en -verfijning te automatiseren. De samenwerkende aard van de techniek zorgt voor een snellere convergentie naar een effectieve prompt, waardoor de tijd en moeite die nodig is voor handmatige promptcreatie wordt verminderd.

Consistentie en Schaalbaarheid: Het gebruik van LLM's in het prompt engineering proces helpt de consistentie tussen prompts te behouden, aangezien de LLM's best practices en patronen kunnen leren en toepassen van eerdere succesvolle prompts. Deze consistentie, gecombineerd met het vermogen om prompts op schaal te genereren, maakt Promptdistillatie een waardevolle techniek voor grootschalige AI-gestuurde applicaties.

Projectidee: Tooling op bibliotheek niveau die het proces van promptversioning en -beoordeling vereenvoudigt in systemen die geautomatiseerde promptdistillaties uitvoeren als onderdeel van hun applicatiecode.

Om Promptdistillatie te implementeren, kunnen ontwikkelaars een workflow of pipeline ontwerpen die LLM's integreert in verschillende fasen van het prompt engineering proces. Dit kan worden bereikt via API-aanroepen, aangepaste tooling of geïntegreerde ontwikkelomgevingen die een naadloze interactie tussen gebruikers en LLM's tijdens promptcreatie mogelijk maken. De specifieke implementatiedetails kunnen variëren afhankelijk van het gekozen LLM-platform en de vereisten van de applicatie.

Hoe zit het met fine-tuning?

In dit boek behandelen we uitgebreid prompt engineering en RAG, maar niet fine-tuning. De belangrijkste reden voor deze beslissing is dat naar mijn mening de meeste applicatieontwikkelaars geen fine-tuning nodig hebben voor hun AI-integratiebehoeften.

Prompt engineering, waarbij zorgvuldig prompts worden opgesteld met zero- tot few-shot voorbeelden, beperkingen en instructies, kan het model effectief begeleiden bij het genereren van relevante en accurate responses voor een breed scala aan taken. Door duidelijke context te bieden en het pad te versmallen via goed ontworpen prompts, kun je de uitgebreide kennis van grote taalmodellen benutten zonder dat fine-tuning nodig is.

Ook Retrieval-Augmented Generation (RAG) biedt een krachtige aanpak voor het integreren van AI in applicaties. Door dynamisch relevante informatie op te halen uit externe kennisbanken of documenten, voorziet RAG het model van gerichte context op het moment van prompting. Dit stelt het model in staat om responses te genereren die nauwkeuriger, actueler en domeinspecifieker zijn, zonder dat het tijd- en resourceintensieve proces van fine-tuning nodig is.

Hoewel fine-tuning nuttig kan zijn voor zeer gespecialiseerde domeinen of taken die een diep niveau van aanpassing vereisen, gaat het vaak gepaard met aanzienlijke rekenkosten, datavereisten en onderhoudsoverhead. Voor de meeste applicatieontwikkelingsscenario's zou de combinatie van effectieve prompt engineering en RAG moeten volstaan om de gewenste AI-gestuurde functionaliteit en gebruikerservaring te bereiken.

Retrieval Augmented Generation (RAG)

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Wat is Retrieval Augmented Generation?

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Hoe werkt RAG?

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Waarom RAG gebruiken in je applicaties?

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

RAG Implementeren in Je Toepassing

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Voorbereiding van Kennisbronnen (Chunking)

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Propositie-chunking

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Implementatie-opmerkingen

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Kwaliteitscontrole

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Voordelen van Propositiegebaseerd Ophalen

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Praktijkvoorbeelden van RAG

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Casestudy: RAG in een Belastingaangifteapplicatie Zonder Embeddings

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Intelligent Query Optimization (IQO)

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Herordening

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

RAG Assessment (RAGAs)

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Betrouwbaarheid

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Antwoordrelevantie

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Contextprecisie

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Contextrelevantie

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Contextherinnering

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Context-entiteitenherinnering

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Answer Semantic Similarity (ANSS)

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Answer Correctness

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Aspect Critique

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Uitdagingen en Toekomstperspectief

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Semantische Segmentatie: Verbetering van Ophaling met Contextbewuste Segmentatie

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Hiërarchische Indexering: Gegevens Structureren voor Verbeterde Retrieval

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Self-RAG: Een Zelfreflectieve Verbetering

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

HyDE: Hypothetical Document Embeddings

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Wat is Contrastief Leren?

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Veelheid aan Werkers

Ik zie mijn AI-componenten graag als kleine, bijna menselijke virtuele “werkers” die naadloos kunnen worden geïntegreerd in mijn applicatielogica om specifieke taken uit te voeren of complexe beslissingen te nemen. Het idee is om de mogelijkheden van het LLM doelbewust te vermenselijken, zodat niemand *te* enthousiast wordt en er magische eigenschappen aan toekent die ze niet bezitten.

In plaats van uitsluitend te vertrouwen op ingewikkelde algoritmen of tijdrovende handmatige implementaties, kunnen ontwikkelaars AI-componenten conceptualiseren als intelligente, toegewijde, mensachtige entiteiten die kunnen worden aangeroepen wanneer nodig om complexe problemen aan te pakken en oplossingen te bieden op basis van hun training en kennis. Deze entiteiten raken niet afgeleid en melden zich niet ziek. Ze besluiten niet spontaan om dingen op een andere manier te doen dan hen is opgedragen, en over het algemeen maken ze, indien correct geprogrammeerd, ook geen fouten.

In technische termen is het belangrijkste principe achter deze aanpak het opdelen van complexe taken of besluitvormingsprocessen in kleinere, beter beheersbare eenheden die door gespecialiseerde AI-werkers kunnen worden afgehandeld. Elke werker is ontworpen om zich te concentreren op een specifiek aspect van het probleem, waarbij deze zijn unieke expertise en mogelijkheden inbrengt. Door de werklast te verdelen over meerdere AI-werkers kan de applicatie een grotere efficiëntie, schaalbaarheid en aanpasbaarheid bereiken.

Neem bijvoorbeeld een webapplicatie die realtime moderatie van door gebruikers gegenereerde content vereist. Het implementeren van een uitgebreid moderatiesysteem vanaf nul zou een ontmoedigende taak zijn, die aanzienlijke ontwikkelingsinspanning en doorlopend onderhoud vereist. Door de Veelheid aan Werkers-aanpak te gebruiken, kunnen ontwikkelaars echter AI-gestuurde moderatiewerkers in de applicatielogica integreren. Deze werkers kunnen automatisch ongepaste inhoud analyseren en markeren, waardoor ontwikkelaars zich kunnen concentreren op andere kritische aspecten van de applicatie.

AI-Werkers Als Onafhankelijke Herbruikbare Componenten

Een belangrijk aspect van de Veelheid aan Werkers-aanpak is de modulariteit. Voorstanders van objectgeoriënteerd programmeren vertellen ons al decennialang dat we over objectinteracties moeten denken als berichten. Wel, AI-werkers kunnen worden ontworpen als onafhankelijke, herbruikbare componenten die via gewone taal met elkaar kunnen “praten”, bijna alsof het echt kleine mensen zijn die met elkaar praten. Deze los gekoppelde aanpak stelt de applicatie in staat zich aan te passen en te evolueren in de loop van de tijd, naarmate nieuwe AI-technologieën ontstaan of bedrijfslogische vereisten veranderen.

In de praktijk is de noodzaak om duidelijke interfaces en communicatieprotocollen

tussen de componenten te ontwerpen niet veranderd, alleen omdat er AI-werkers bij betrokken zijn. Je moet nog steeds rekening houden met andere factoren zoals prestaties, schaalbaarheid en beveiliging, maar nu zijn er ook volledig nieuwe “zachte vereisten” om rekening mee te houden. Veel gebruikers hebben bijvoorbeeld bezwaar tegen het gebruik van hun privégegevens voor het trainen van nieuwe AI-modellen. Heb je het privacyniveau geverifieerd dat wordt geboden door de modelprovider die je gebruikt?

AI-Werkers Als Microservices?

Als je leest over de Veelheid aan Werkers-aanpak, zul je misschien enkele overeenkomsten opmerken met Microservices-architectuur. Beide benadrukken de ontleding van complexe systemen in kleinere, beter beheersbare en onafhankelijk inzetbare eenheden. Net zoals microservices zijn ontworpen om los gekoppeld te zijn, gericht op specifieke bedrijfsmogelijkheden en te communiceren via goed gedefinieerde API’s, zijn AI-werkers ontworpen om modulair te zijn, gespecialiseerd in hun taken en met elkaar te interacteren via duidelijke interfaces en communicatieprotocollen.

Er zijn echter enkele belangrijke verschillen om rekening mee te houden. Terwijl microservices typisch worden geïmplementeerd als afzonderlijke processen of services die op verschillende machines of containers draaien, kunnen AI-werkers worden geïmplementeerd als zelfstandige componenten binnen een enkele applicatie of als afzonderlijke services, afhankelijk van je specifieke vereisten en schaalbaarheidsbehoeften. Bovendien omvat de communicatie tussen AI-werkers vaak het uitwisselen van rijke, op natuurlijke taal gebaseerde informatie, zoals prompts, instructies en gegenereerde content, in plaats van de meer gestructureerde dataformaten die gewoonlijk in microservices worden gebruikt.

Ondanks deze verschillen blijven de principes van modulariteit, losse koppeling en duidelijke communicatie-interfaces centraal staan in beide patronen. Door deze

> principes toe te passen op je AI-werkerarchitectuur, kun je flexibele, schaalbare en onderhoudbare systemen creëren die de kracht van AI benutten om complexe problemen op te lossen en waarde te leveren aan je gebruikers.

De Veelheid aan Werkers-aanpak kan worden toegepast in verschillende domeinen en applicaties, waarbij de kracht van AI wordt benut om complexe taken aan te pakken en intelligente oplossingen te leveren. Laten we enkele concrete voorbeelden bekijken van hoe AI-werkers kunnen worden ingezet in verschillende contexten.

Accountbeheer

Praktisch elke zelfstandige webapplicatie heeft het concept van een account (of gebruiker). In Olympia gebruiken we een `AccountManager` AI-werker die is geprogrammeerd om verschillende soorten wijzigingsverzoeken met betrekking tot gebruikersaccounts te kunnen afhandelen.

De richtlijn luidt als volgt:

```
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.
```

De initiële staat van het account geproduceerd door `account.to_directive` is simpelweg een tekstuele beschrijving van het account, inclusief relevante gerelateerde gegevens zoals gebruikers, abonnementen, etc.

Het scala aan functies beschikbaar voor de `AccountManager` geeft het de mogelijkheid om het abonnement van de gebruiker te bewerken, AI-consultants en andere betaalde add-ons toe te voegen en te verwijderen, en notificatie-e-mails te versturen naar de accounteigenaar. Naast de `finished` functie kan het ook `notify_human_administrator` aanroepen als het een fout tegenkomt tijdens de verwerking of andere vorm van assistentie nodig heeft bij een verzoek.

Merk op dat in het geval van vragen, de `AccountManager` ervoor kan kiezen om Olympia's kennisbank te doorzoeken, waar het instructies kan vinden over hoe om te gaan met randgevallen en andere situaties waarbij het niet zeker is hoe verder te gaan.

E-commerce Toepassingen

In de wereld van e-commerce kunnen AI-medewerkers een cruciale rol spelen bij het verbeteren van de gebruikerservaring en het optimaliseren van bedrijfsprocessen. Hier zijn enkele manieren waarop AI-medewerkers kunnen worden ingezet:

Productaanbevelingen

Een van de krachtigste toepassingen van AI-medewerkers in e-commerce is het genereren van gepersonaliseerde productaanbevelingen. Door het analyseren van gebruikersgedrag, aankoopgeschiedenis en voorkeuren kunnen deze medewerkers producten suggereren die zijn afgestemd op de interesses en behoeften van elke individuele gebruiker.

De sleutel tot effectieve productaanbevelingen is het benutten van een combinatie van collaboratieve filtering en op inhoud gebaseerde filtering technieken. Collaboratieve

filtering kijkt naar het gedrag van vergelijkbare gebruikers om patronen te identificeren en aanbevelingen te doen op basis van wat anderen met vergelijkbare voorkeuren hebben gekocht of waardevol vonden. Op inhoud gebaseerde filtering richt zich daarentegen op de kenmerken en eigenschappen van de producten zelf, waarbij items worden aanbevolen die vergelijkbare eigenschappen delen met producten waarin een gebruiker eerder interesse heeft getoond.

Hier is een vereenvoudigd voorbeeld van hoe je een productaanbevelingsmedewerker in Ruby kunt implementeren, dit keer met gebruik van een "Railway Oriented (ROP)" functionele programmeerstijl:

```
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
```

De stijl van Ruby functioneel programmeren die in het voorbeeld wordt gebruikt, is beïnvloed door F# en Rust. Je kunt er meer over lezen in de uitleg van de techniek van mijn vriend Chad Wooley bij GitLab

In dit voorbeeld neemt de `ProductRecommendationWorker` een gebruiker als invoer

en genereert gepersonaliseerde productaanbevelingen door een waardeobject door een keten van functionele stappen te leiden. Laten we elke stap ontleden:

1. `ValidateUser.validate`: Deze stap zorgt ervoor dat de gebruiker geldig is en in aanmerking komt voor gepersonaliseerde aanbevelingen. Het controleert of de gebruiker bestaat, actief is en of de nodige gegevens beschikbaar zijn voor het genereren van aanbevelingen. Als de validatie mislukt, wordt een foutresultaat teruggegeven en wordt de keten vroegtijdig afgebroken.
2. `AnalyzeCurrentSession.analyze`: Als de gebruiker geldig is, analyseert deze stap de huidige browsersessie van de gebruiker om contextuele informatie te verzamelen. Het kijkt naar recente interacties van de gebruiker, zoals bekeken producten, zoekopdrachten en winkelwagentinhoud, om hun huidige interesses en intenties te begrijpen.
3. `CollaborativeFilter.filter`: Met behulp van het *gedrag van vergelijkbare gebruikers*, past deze stap collaboratieve filteringtechnieken toe om producten te identificeren die waarschijnlijk interessant zijn voor de gebruiker. Het houdt rekening met factoren zoals aankoopgeschiedenis, beoordelingen en gebruiker-item-interacties om een set van kandidaat-aanbevelingen te genereren.
4. `ContentBasedFilter.filter`: Deze stap verfijnt de kandidaat-aanbevelingen verder door op inhoud gebaseerde filtering toe te passen. Het vergelijkt de attributen en kenmerken van de kandidaat-producten met de *voorkeuren en historische gegevens van de gebruiker* om de meest relevante items te selecteren.
5. `ProductSelector.select`: Ten slotte selecteert deze stap de top N producten uit de gefilterde aanbevelingen op basis van vooraf gedefinieerde criteria, zoals relevantiescore, populariteit of andere bedrijfsregels. De geselecteerde producten worden vervolgens teruggegeven als de definitieve gepersonaliseerde aanbevelingen.

De schoonheid van het gebruik van een functionele Ruby programmeerstijl hier is dat het ons in staat stelt deze stappen op een heldere en beknopte manier aan elkaar te

ketenen. Elke stap richt zich op een specifieke taak en geeft een `Result` object terug, dat ofwel een succes (`ok`) of een fout (`err`) kan zijn. Als een stap een fout tegenkomt, wordt de keten vroegtijdig afgebroken en wordt de fout doorgegeven aan het eindresultaat.

In de `case` statement aan het einde doen we aan pattern matching op het eindresultaat. Als het resultaat een fout is (`ProductRecommendationError`), loggen we de fout met behulp van een tool zoals Honeybadger voor monitoring en debugging doeleinden. Als het resultaat een succes is (`ProductRecommendations`), zenden we een `:new_recommendations` event uit met behulp van de Wisper pub/sub bibliotheek, waarbij we de gebruiker en de gegenereerde aanbevelingen meegeven.

Door gebruik te maken van functionele programmeertechnieken kunnen we een modulaire en onderhoudbare product aanbevelingsworker creëren. Elke stap is op zichzelf staand en kan eenvoudig worden getest, aangepast of vervangen zonder de algemene flow te beïnvloeden. Het gebruik van pattern matching en de `Result` klasse helpt ons om fouten netjes af te handelen en zorgt ervoor dat de worker snel faalt als een stap een probleem tegenkomt.

Dit is natuurlijk een vereenvoudigd voorbeeld, en in een echte situatie zou je moeten integreren met je e-commerce platform, randgevallen moeten afhandelen, en zelfs de implementatie van de aanbevelingsalgoritmen moeten aanpakken. Echter blijven de kernprincipes van het opdelen van het probleem in kleinere stappen en het benutten van functionele programmeertechnieken hetzelfde.

Fraudedetectie

Hier is een vereenvoudigd voorbeeld van hoe je een fraudedetectie worker kunt implementeren met dezelfde Railway Oriented Programming (ROP) stijl in 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
```

De `FraudDetection` klasse is een *value object* dat de fraudedetectiestatus voor een gegeven transactie inkapselt. Het biedt een gestructureerde manier om het frauderisico van een transactie te analyseren en te beoordelen op basis van verschillende risicofactoren.

```ruby
class FraudDetection
  RISK_THRESHOLD = 0.8

  attr_accessor :transaction, :risk_factors

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

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

  def high_risk?
    fraud_probability > RISK_THRESHOLD
  end

  private

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

De `FraudDetection` klasse heeft de volgende attributen:

- `transaction`: Een referentie naar de transactie die wordt geanalyseerd op fraude.
- `risk_factors`: Een array die de risicofactoren van de transactie opslaat. Elke risicofactor wordt weergegeven als een hash, waarbij de sleutel de beschrijving van de risicofactor is en de waarde de fraudewaarschijnlijkheid die bij die risicofactor hoort.

De `add_risk_factor` methode maakt het mogelijk om een risicofactor toe te voegen

aan de `risk_factors` array. Deze methode heeft twee parameters: `description`, wat een string is die de risicofactor beschrijft, en `probability`, wat een float is die de fraudewaarschijnlijkheid van die risicofactor weergeeft. We gebruiken een `case..in` voorwaarde om een eenvoudige typecontrole uit te voeren.

De `high_risk?` methode die aan het einde van de keten wordt gecontroleerd, is een predicaatmethode die de `fraud_probability` (berekend door de som van alle risicofactorwaarschijnlijkheden) vergelijkt met de `RISK_THRESHOLD`.

De `FraudDetection` klasse biedt een nette en ingekapselde manier om fraudedetectie voor een transactie te beheren. Het maakt het mogelijk om meerdere risicofactoren toe te voegen, elk met een eigen beschrijving en waarschijnlijkheid, en biedt een methode om te bepalen of de transactie als hoog risico wordt beschouwd op basis van de berekende fraudewaarschijnlijkheid. De klasse kan eenvoudig worden geïntegreerd in een groter fraudedetectiesysteem, waarbij verschillende componenten kunnen samenwerken om het risico op frauduleuze transacties te beoordelen en te beperken.

Tot slot, aangezien dit tenslotte een boek is over programmeren met AI, hier is een voorbeeldimplementatie van de `CheckCustomerHistory` klasse die gebruik maakt van AI-verwerking met behulp van mijn Raix bibliotheek's `ChatCompletion` module:

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

In dit voorbeeld definieert de `CheckCustomerHistory` een `INSTRUCTION` constante die specifieke instructies geeft aan het AI-model over hoe het de transactiegeschiedenis van de klant moet analyseren op mogelijke fraude-indicatoren via een systeemrichtlijn

De `self.check` methode is een klassemethode die een nieuwe instantie van

`CheckCustomerHistory` initialiseert met het `fraud_detection` object en de `call` methode aanroept om de klantgeschiedenisanalyse uit te voeren.

In de `call` methode wordt de transactiegeschiedenis van de klant opgehaald en geformatteerd in een transcript dat naar het AI-model wordt gestuurd. Het AI-model analyseert de transactiegeschiedenis op basis van de gegeven instructies en geeft een samenvatting van zijn bevindingen terug.

De bevindingen worden toegevoegd aan het `fraud_detection` object, en het bijgewerkte `fraud_detection` object wordt teruggegeven als een succesvolle `Result`.

Door gebruik te maken van de `ChatCompletion` module kan de `CheckCustomerHistory` klasse de kracht van AI benutten om de transactiegeschiedenis van de klant te analyseren en mogelijke fraude-indicatoren te identificeren. Dit maakt meer geavanceerde en adaptieve fraudedetectietechnieken mogelijk, aangezien het AI-model nieuwe patronen en anomalieën kan leren en zich daaraan kan aanpassen.

De bijgewerkte `FraudDetectionWorker` en de `CheckCustomerHistory` klasse laten zien hoe AI-workers naadloos kunnen worden geïntegreerd, waardoor het fraudedetectieproces wordt versterkt met intelligente analyse- en besluitvormingscapaciteiten.

Klantsentimentanalyse

Hier is nog een vergelijkbaar voorbeeld van hoe je een klantsentimentanalyse-worker kunt implementeren. Deze keer met veel minder uitleg, aangezien je nu wel begrijpt hoe deze programmeerstijl werkt:

```
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
```

In dit voorbeeld omvatten de stappen van de `CustomerSentimentAnalysisWorker` het voorbewerken van de feedback (bijvoorbeeld het verwijderen van ruis, tokenisatie), het uitvoeren van sentimentanalyse om het algemene sentiment te bepalen (positief, negatief of neutraal), het extraheren van belangrijke zinsneden en onderwerpen, het identificeren van trends en patronen, en het genereren van bruikbare inzichten op basis van de analyse.

Toepassingen in de Gezondheidszorg

In de gezondheidszorg kunnen AI-workers medische professionals en onderzoekers ondersteunen bij verschillende taken, wat leidt tot verbeterde patiëntresultaten en versnelde medische ontdekkingen. Enkele voorbeelden zijn:

Patiëntenregistratie

AI-workers kunnen het patiëntenregistratieproces stroomlijnen door verschillende taken te automatiseren en intelligente ondersteuning te bieden.

Afsprakenplanning: AI-workers kunnen afsprakenplanning afhandelen door rekening te houden met patiëntvoorkeuren, beschikbaarheid en de urgentie van hun medische behoeften. Ze kunnen communiceren met patiënten via gespreksinterfaces, hen begeleiden door het planningsproces en de meest geschikte afspraaktijden vinden op basis van de vereisten van de patiënt en de beschikbaarheid van de zorgverlener.

Verzameling van Medische Voorgeschiedenis: Tijdens de patiëntenregistratie kunnen AI-workers helpen bij het verzamelen en documenteren van de medische voorgeschiedenis van de patiënt. Ze kunnen interactieve gesprekken voeren met patiënten, relevante vragen stellen over hun eerdere medische aandoeningen, medicatie, allergieën en familiegeschiedenis. De AI-workers kunnen natuurlijke taalverwerkingstechnieken gebruiken om de verzamelde informatie te interpreteren en te structureren, waarbij wordt gezorgd dat deze nauwkeurig wordt vastgelegd in het elektronische patiëntendossier.

Symptoombeoordeeling en Stratificatie: AI-workers kunnen initiële symptoombeoordelingen uitvoeren door patiënten te vragen naar hun huidige symptomen, duur, ernst en eventuele gerelateerde factoren. Door gebruik te maken van medische kennisbanken en machine learning-modellen kunnen deze workers de verstrekte informatie analyseren en voorlopige differentiële diagnoses genereren of passende vervolgstappen aanbevelen, zoals het plannen van een consult met een zorgverlener of het voorstellen van zelfzorgmaatregelen.

Verzekeringverificatie: AI-workers kunnen helpen bij verzekeringverificatie tijdens de patiëntenregistratie. Ze kunnen patiëntverzekeringgegevens verzamelen, communiceren met verzekeraars via API's of webservices, en de dekkingsgeschiktheid en uitkeringen verifiëren. Deze automatisering helpt het verzekeringverificatieproces

te stroomlijnen, vermindert de administratieve last en zorgt voor nauwkeurige informatievastlegging.

Patiëntenvoorlichting en Instructies: AI-workers kunnen patiënten voorzien van relevante voorlichtingsmaterialen en instructies op basis van hun specifieke medische aandoeningen of aankomende procedures. Ze kunnen gepersonaliseerde inhoud leveren, veelgestelde vragen beantwoorden en begeleiding bieden bij voorbereidingen voor afspraken, medicatie-instructies of nazorg. Dit helpt patiënten geïnformeerd en betrokken te houden gedurende hun zorgtraject.

Door AI-workers in te zetten bij patiëntenregistratie kunnen zorginstellingen de efficiëntie verhogen, wachttijden verkorten en de algemene patiëntervaring verbeteren. Deze workers kunnen routinematige taken afhandelen, nauwkeurige informatie verzamelen en gepersonaliseerde ondersteuning bieden, waardoor zorgprofessionals zich kunnen concentreren op het leveren van hoogwaardige zorg aan patiënten.

Patiëntrisicobeoordeling

AI-workers kunnen een cruciale rol spelen bij het beoordelen van patiëntrisico's door verschillende gegevensbronnen te analyseren en geavanceerde analysetechnieken toe te passen.

Gegevensintegratie: AI-workers kunnen patiëntgegevens uit verschillende bronnen verzamelen en interpreteren, zoals elektronische patiëntendossiers (EPD's), medische beeldvorming, laboratoriumresultaten, draagbare apparaten en sociale gezondheidsdeterminanten. Door deze informatie te consolideren in een uitgebreid patiëntprofiel kunnen AI-workers een holistisch beeld geven van de gezondheidstoestand en risicofactoren van de patiënt.

Risicostratificatie: AI-workers kunnen voorspellende modellen gebruiken om patiënten in verschillende risicocategorieën in te delen op basis van hun individuele kenmerken en gezondheidsgegevens. Deze risicostratificatie stelt zorgverleners in staat

prioriteit te geven aan patiënten die meer directe aandacht of interventie nodig hebben. Bijvoorbeeld, patiënten die als hoogrisico voor een bepaalde aandoening worden geïdentificeerd, kunnen worden gemarkeerd voor nauwlettender toezicht, preventieve maatregelen of vroege interventie.

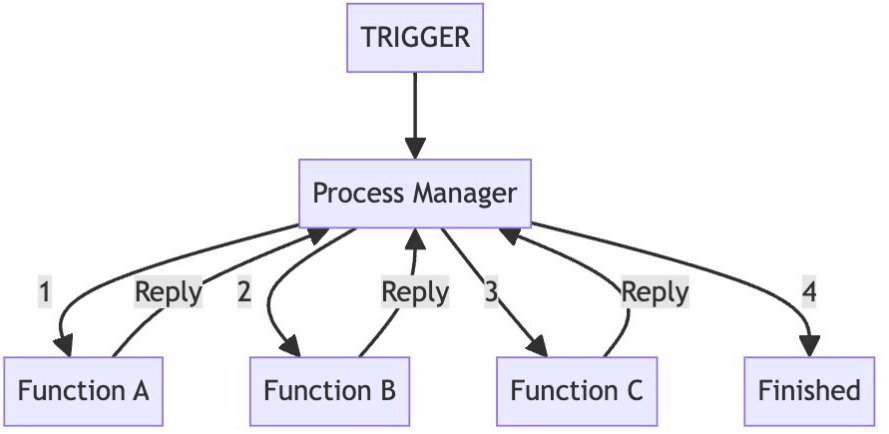
Gepersonaliseerde Risicoprofielen: AI-workers kunnen gepersonaliseerde risicoprofielen genereren voor elke patiënt, waarbij de specifieke factoren die bijdragen aan hun risicoscores worden belicht. Deze profielen kunnen inzichten bevatten in de levensstijl van de patiënt, genetische aanleg, omgevingsfactoren en sociale gezondheidsdeterminanten. Door een gedetailleerde uitsplitsing van risicofactoren te geven, kunnen AI-workers zorgverleners helpen preventiestrategieën en behandelplannen af te stemmen op individuele patiëntbehoeften.

Continue Risicobewaking: AI-workers kunnen patiëntgegevens continu monitoren en risicobeoordelingen in realtime bijwerken. Wanneer nieuwe informatie beschikbaar komt, zoals veranderingen in vitale functies, laboratoriumresultaten of therapietrouw, kunnen AI-workers risicoscores herberekenen en zorgverleners waarschuwen voor significante veranderingen. Deze proactieve monitoring maakt tijdige interventies en aanpassingen in patiëntzorgplannen mogelijk.

Klinische Beslissingsondersteuning: AI-workers kunnen resultaten van risicobeoordelingen integreren in klinische beslissingsondersteunende systemen, waarbij zorgverleners worden voorzien van evidence-based aanbevelingen en waarschuwingen. Als bijvoorbeeld de risicoscore van een patiënt voor een bepaalde aandoening een bepaalde drempel overschrijdt, kan de AI-worker de zorgverlener aansporen om specifieke diagnostische tests, preventieve maatregelen of behandelingsopties te overwegen op basis van klinische richtlijnen en best practices.

Deze workers kunnen enorme hoeveelheden patiëntgegevens verwerken, geavanceerde analyses toepassen en bruikbare inzichten genereren ter ondersteuning van klinische besluitvorming. Dit leidt uiteindelijk tot verbeterde patiëntresultaten, lagere zorgkosten en beter populatiegezondheidsmanagement.

AI Worker als Procesmanager



In de context van AI-gestuurde applicaties kan een worker worden ontworpen om te functioneren als een Procesmanager, zoals beschreven in het boek “Enterprise Integration Patterns” van Gregor Hohpe. Een Procesmanager is een centraal component dat de status van een proces bijhoudt en de volgende verwerkingsstappen bepaalt op basis van tussenresultaten.

Wanneer een AI-worker als Procesmanager functioneert, ontvangt deze een inkomend bericht dat het proces initialiseert, bekend als het *triggebericht*. De AI-worker houdt vervolgens de status van de procesuitvoering bij (als een conversatieverslag) en verwerkt het bericht via een reeks verwerkingsstappen die zijn geïmplementeerd als toolfuncties, die sequentieel of parallel kunnen worden aangeroepen, naar eigen inzicht.

Als je een AI-modelklasse zoals GPT-4 gebruikt die weet hoe je functies parallel moet uitvoeren, dan kan je worker meerdere stappen tegelijkertijd uitvoeren. Eerlijk gezegd heb ik dit zelf niet geprobeerd en mijn gevoel zegt dat je resultaten kunnen variëren.

Na elke individuele verwerkingsstap wordt de controle teruggegeven aan de AI-worker, waardoor deze de volgende verwerkingsstap(pen) kan bepalen op basis van de huidige status en de verkregen resultaten.

Sla Je Triggeberichten Op

Volgens mijn ervaring is het verstandig om je triggebericht te implementeren als een database-ondersteund object. Op die manier wordt elke procesinstantie geïdentificeerd door een unieke primaire sleutel en heb je een plek om de status van de uitvoering op te slaan, inclusief het conversatieverslag van de AI.

Hier is bijvoorbeeld een vereenvoudigde versie van Olympia's `AccountChange` modelklasse, die een verzoek vertegenwoordigt om een wijziging aan te brengen in het account van een gebruiker.

```
# == 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
```

```
23
24    validates :description, presence: true
25
26    after_commit -> {
27      broadcast(:account_change_requested, self)
28    }, on: :create
29
30    state_machine initial: :requested do
31      event :completed do
32        transition all => :complete
33      end
34      event :failed do
35        transition all => :requires_human_review
36      end
37    end
38  end
```

De `AccountChange` klasse fungeert als een triggerbericht dat een proces start om het verzoek tot accountwijziging af te handelen. Merk op hoe het wordt uitgezonden naar Olympia's Wisper-gebaseerde pub/sub subsysteem nadat de create-transactie is voltooid.

Het opslaan van het triggerbericht in de database op deze manier zorgt voor een blijvende registratie van elk verzoek tot accountwijziging. Elk exemplaar van de `AccountChange` klasse krijgt een unieke primaire sleutel toegewezen, wat een eenvoudige identificatie en tracking van individuele verzoeken mogelijk maakt. Dit is vooral nuttig voor auditlogboekregistratie, omdat het systeem hiermee een historische registratie kan bijhouden van alle accountwijzigingen, inclusief wanneer ze werden aangevraagd, welke wijzigingen werden verzocht en de huidige status van elk verzoek.

In het gegeven voorbeeld bevat de `AccountChange` klasse velden zoals `description` om de details van de gevraagde wijziging vast te leggen, `state` om de huidige status van het verzoek weer te geven (bijvoorbeeld requested, complete, requires_human_review), en `transcript` om het AI-gesprekstranscript gerelateerd aan het verzoek op te slaan. Het `description` veld is de daadwerkelijke prompt die wordt gebruikt om de eerste chatcompletion met de AI te initiëren. Het opslaan van deze gegevens biedt waardevolle

context en maakt betere tracking en analyse van het accountwijzigingsproces mogelijk.

Het opslaan van triggerberichten in de database maakt robuuste foutafhandeling en herstel mogelijk. Als er een fout optreedt tijdens de verwerking van een accountwijzigingsverzoek, markeert het systeem het verzoek als mislukt en zet het over naar een status die menselijke interventie vereist. Dit zorgt ervoor dat geen enkel verzoek verloren gaat of wordt vergeten, en dat eventuele problemen correct kunnen worden aangepakt en opgelost.

De AI-worker, als Procesmanager, biedt een centraal controlepunt en maakt krachtige procesrapportage en debugging-mogelijkheden mogelijk. Het is echter belangrijk op te merken dat het gebruik van een AI-worker als Procesmanager voor elk werkstroomscenario in uw applicatie mogelijk overdreven is.

AI-Workers Integreren in Uw Applicatiearchitectuur

Bij het integreren van AI-workers in uw applicatiearchitectuur moeten verschillende technische overwegingen worden aangepakt om een soepele integratie en effectieve communicatie tussen de AI-workers en andere applicatiecomponenten te waarborgen. Deze sectie behandelt belangrijke aspecten van het ontwerpen van die interfaces, het afhandelen van gegevensstromen en het beheren van de levenscyclus van AI-workers.

Het Ontwerpen van Duidelijke Interfaces en Communicatieprotocollen

Om een naadloze integratie tussen AI-workers en andere applicatiecomponenten te faciliteren, is het cruciaal om duidelijke interfaces en communicatieprotocollen te definiëren. Overweeg de volgende benaderingen:

API-gebaseerde Integratie: Stel de functionaliteit van AI-workers beschikbaar via goed gedefinieerde API's, zoals RESTful endpoints of GraphQL schema's. Dit stelt andere componenten in staat om met de AI-workers te communiceren via standaard HTTP-verzoeken en -responses. API-gebaseerde integratie biedt een duidelijk contract tussen de AI-workers en de consumerende componenten, waardoor het ontwikkelen, testen en onderhouden van de integratiepunten eenvoudiger wordt.

Berichtgebaseerde Communicatie: Implementeer berichtgebaseerde communicatiepatronen, zoals message queues of publiceer-abonneer systemen, om asynchrone interactie tussen AI-workers en andere componenten mogelijk te maken. Deze aanpak ontkoppelt de AI-workers van de rest van de applicatie, wat zorgt voor betere schaalbaarheid, fouttolerantie en losse koppeling. Berichtgebaseerde communicatie is vooral nuttig wanneer de verwerking door AI-workers tijdrovend of resource-intensief is, omdat het andere delen van de applicatie in staat stelt door te gaan met uitvoeren zonder te wachten tot de AI-workers hun taken hebben voltooid.

Gebeurtenisgestuurde Architectuur: Ontwerp uw systeem rond gebeurtenissen en triggers die AI-workers activeren wanneer aan specifieke voorwaarden wordt voldaan. AI-workers kunnen zich abonneren op relevante gebeurtenissen en daarop reageren door hun aangewezen taken uit te voeren wanneer de gebeurtenissen plaatsvinden. Gebeurtenisgestuurde architectuur maakt realtime verwerking mogelijk en zorgt ervoor dat AI-workers op aanvraag kunnen worden aangeroepen, waardoor onnodig resourceverbruik wordt verminderd. Deze aanpak is zeer geschikt voor scenario's waarbij AI-workers moeten reageren op specifieke acties of veranderingen in de applicatiestatus.

Gegevensstroom en Synchronisatie Afhandelen

Bij het integreren van AI-workers in uw applicatie is het cruciaal om een soepele gegevensstroom te waarborgen en gegevensconsistentie te behouden tussen de AI-workers en andere componenten. Overweeg de volgende aspecten:

Gegevensvoorbereiding: Voordat gegevens worden ingevoerd in AI-workers, moet u mogelijk verschillende gegevensvoorbereidingstaken uitvoeren, zoals het opschonen, formatteren en/of transformeren van de invoergegevens. U wilt niet alleen zorgen dat de AI-workers effectief kunnen verwerken, maar ook dat u geen tokens verspilt door aandacht te besteden aan informatie die de worker in het beste geval nutteloos en in het slechtste geval afleidend kan vinden. Gegevensvoorbereiding kan taken omvatten zoals het verwijderen van ruis, het afhandelen van ontbrekende waarden of het converteren van gegevenstypen.

Gegevenspersistentie: Hoe gaat u de gegevens die in en uit AI-workers stromen opslaan en persisteren? Overweeg factoren zoals gegevensvolume, querypatronen en schaalbaarheid. Moet u het AI-transcript bewaren als een weerspiegeling van het "denkproces" voor audit- of debuggingdoeleinden, of volstaat het om alleen een registratie van de resultaten te hebben?

Gegevensophaling: Het verkrijgen van de benodigde gegevens door workers kan bestaan uit het bevragen van databases, het lezen van bestanden of het benaderen van externe API's. Houd rekening met latentie en hoe AI-workers toegang krijgen tot de meest actuele gegevens. Hebben ze volledige toegang tot uw database nodig of moet u de reikwijdte van hun toegang nauw definiëren op basis van hun taken? En hoe zit het met schaalbaarheid? Overweeg cachingmechanismen om de prestaties te verbeteren en de belasting van de onderliggende gegevensbronnen te verminderen.

Gegevenssynchronisatie: Wanneer meerdere componenten, inclusief AI-workers, gedeelde gegevens benaderen en wijzigen, is het belangrijk om geschikte synchronisatiemechanismen te implementeren om gegevensconsistentie te waarborgen. Database vergrendelingsstrategieën, zoals optimistische of pessimistische vergrendeling, kunnen u helpen conflicten te voorkomen en gegevensintegriteit te waarborgen. Implementeer transactiebeheertechnieken om gerelateerde gegevensbewerkingen te groeperen en de ACID-eigenschappen (atomiciteit, consistentie, isolatie en duurzaamheid) te behouden.

Foutafhandeling en Herstel: Implementeer robuuste foutafhandelings- en herstelmechanismen om gegevensgerelateerde problemen aan te pakken die kunnen ontstaan tijdens het gegevensstroomproces. Handel uitzonderingen netjes af en voorzie in betekenisvolle foutmeldingen om te helpen bij het debuggen. Implementeer hertrypogingsmechanismen en terugvalmethoden om tijdelijke storingen of netwerkonderbrekingen af te handelen. Definieer duidelijke procedures voor gegevensherstel en -restauratie in geval van gegevensbeschadiging of -verlies.

Door zorgvuldig gegevensstromen en synchronisatiemechanismen te ontwerpen en implementeren, kunt u ervoor zorgen dat uw AI-workers toegang hebben tot nauwkeurige, consistente en actuele gegevens. Dit stelt hen in staat om hun taken effectief uit te voeren en betrouwbare resultaten te produceren.

Beheer van de Levenscyclus van AI-Workers

Ontwikkel een gestandaardiseerd proces voor het initialiseren en configureren van AI-workers. Ik heb een voorkeur voor frameworks die standaardiseren hoe je instellingen definieert zoals modelnamen, systeemrichtlijnen en functiedefinities. Zorg ervoor dat het initialisatieproces geautomatiseerd en reproduceerbaar is om implementatie en schaalbaarheid te vergemakkelijken.

Implementeer uitgebreide monitoring- en loggingmechanismen om de gezondheid en prestaties van AI-workers te volgen. Verzamel metrics zoals brongebruik, verwerkingstijd, foutpercentages, en doorvoer. Gebruik gecentraliseerde loggingsystemen zoals ELK stack (Elasticsearch, Logstash, Kibana) om logs van meerdere AI-workers te verzamelen en analyseren.

Bouw fouttolerantie en veerkracht in de AI-worker architectuur. Implementeer foutafhandelings- en herstelmechanismen om storingen of uitzonderingen elegant af te handelen. Grote Taalmodellen zijn nog steeds zeer nieuwe technologie; providers vallen vaak onverwacht uit. Gebruik hertrypogingsmechanismen en stroomonderbrekers om cascade-effecten bij storingen te voorkomen.

Samenstelbaarheid en Orchestratie van AI-Workers

Een van de belangrijkste voordelen van de AI-worker architectuur is de samenstelbaarheid, waardoor je meerdere AI-workers kunt combineren en orchestreren om complexe problemen op te lossen. Door een grotere taak op te splitsen in kleinere, beter beheersbare subtaken, elk behandeld door een gespecialiseerde AI-worker, kun je krachtige en flexibele systemen creëren. In deze sectie verkennen we verschillende benaderingen voor het samenstellen en orchestreren van “een veelvoud” aan AI-workers.

AI-Workers Aaneenschakelen voor Meerstaps Werkstromen

In veel scenario’s kan een complexe taak worden opgedeeld in een reeks opeenvolgende stappen, waarbij de output van één AI-worker de input wordt voor de volgende. Deze aaneenschakeling van AI-workers creëert een meerstaps werkstroom of pipeline. Elke AI-worker in de keten richt zich op een specifieke subtaak, en de uiteindelijke output is het resultaat van de gecombineerde inspanningen van alle workers.

Laten we een voorbeeld bekijken in de context van een Ruby on Rails-applicatie voor het verwerken van door gebruikers gegenereerde content. De werkstroom omvat de volgende stappen, die toegegeven waarschijnlijk elk te eenvoudig zijn om in praktijksituaties op deze manier op te delen, maar ze maken het voorbeeld makkelijker te begrijpen:

1. **Tekstopschoning**: Een AI-worker verantwoordelijk voor het verwijderen van HTML-tags, het omzetten van tekst naar kleine letters en het afhandelen van Unicode-normalisatie.

2. **Taaldetectie**: Een AI-worker die de taal van de opgeschoonde tekst identificeert.

3. **Sentimentanalyse**: Een AI-worker die het sentiment (positief, negatief of neutraal) van de tekst bepaalt op basis van de gedetecteerde taal.

4. **Contentcategorisering**: Een AI-worker die de tekst classificeert in voorgedefinieerde categorieën met behulp van natuurlijke taalverwerkingstechnieken.

Hier is een zeer vereenvoudigd voorbeeld van hoe je deze AI-workers aan elkaar kunt schakelen met behulp van Ruby:

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

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

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

In dit voorbeeld initialiseert de `ContentProcessor` klasse met de ruwe tekst en schakelt de AI-workers aan elkaar in de `process` methode. Elke AI-worker voert zijn specifieke taak uit en geeft het resultaat door aan de volgende worker in de keten. De uiteindelijke output is een hash die de opgeschoonde tekst, gedetecteerde taal, sentiment en contentcategorie bevat.

Parallelle Verwerking voor Onafhankelijke AI-Workers

In het vorige voorbeeld zijn de AI-workers sequentieel geschakeld, waarbij elke worker de tekst verwerkt en het resultaat doorgeeft aan de volgende worker. Echter, als je meerdere AI-workers hebt die onafhankelijk van elkaar op dezelfde input kunnen werken, kun je de workflow optimaliseren door ze parallel te verwerken.

In het gegeven scenario kan, zodra de tekstopschoning is uitgevoerd door de `TextCleanupWorker`, de `LanguageDetectionWorker`, `SentimentAnalysisWorker`, en `CategorizationWorker` allemaal onafhankelijk de opgeschoonde tekst verwerken. Door deze workers parallel uit te voeren, kun je mogelijk de totale verwerkingstijd verminderen en de efficiëntie van je workflow verbeteren.

Om parallelle verwerking in Ruby te bereiken, kun je gebruik maken van gelijktijdigheidstechnieken zoals threads of asynchrone programmering. Hier is een voorbeeld van hoe je de `ContentProcessor` klasse kunt aanpassen om de laatste drie workers parallel te verwerken met behulp van threads:

```
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
```

```
29 end
```

In deze geoptimaliseerde versie gebruiken we de `concurrent-ruby` bibliotheek om `Concurrent::Future` objecten te maken voor elk van de onafhankelijke AI-workers. Een `Future` representeert een berekening die asynchroon wordt uitgevoerd in een aparte thread.

Na de tekstopschoningsstap maken we drie `Future` objecten: `language_future`, `sentiment_future`, en `category_future`. Elke `Future` voert zijn corresponderende AI-worker uit (`LanguageDetectionWorker`, `SentimentAnalysisWorker`, en `CategorizationWorker`) in een aparte thread, waarbij de `cleaned_text` als invoer wordt doorgegeven.

Door de `value` methode aan te roepen op elke `Future`, wachten we tot de berekening voltooid is en halen we het resultaat op. De `value` methode blokkeert totdat het resultaat beschikbaar is, wat ervoor zorgt dat alle parallelle workers klaar zijn met verwerken voordat we verdergaan.

Tot slot bouwen we de output hash met de opgeschoonde tekst en de resultaten van de parallelle workers, net zoals in het originele voorbeeld.

Door de onafhankelijke AI-workers parallel te verwerken, kun je mogelijk de totale verwerkingstijd verminderen in vergelijking met sequentiële uitvoering. Deze optimalisatie is vooral voordelig bij tijdrovende taken of bij het verwerken van grote hoeveelheden data.

Het is echter belangrijk op te merken dat de daadwerkelijke prestatiewinst afhangt van verschillende factoren, zoals de complexiteit van elke worker, de beschikbare systeembronnen en de overhead van thread-beheer. Het is altijd een goede gewoonte om je code te benchmarken en te profileren om het optimale niveau van parallellisme voor jouw specifieke gebruik te bepalen.

Daarnaast moet je bij het implementeren van parallelle verwerking rekening houden met eventuele gedeelde bronnen of afhankelijkheden tussen de workers. Zorg ervoor

dat de workers onafhankelijk kunnen opereren zonder conflicten of race conditions. Als er afhankelijkheden of gedeelde bronnen zijn, moet je mogelijk geschikte synchronisatiemechanismen implementeren om data-integriteit te behouden en problemen zoals deadlocks of inconsistente resultaten te voorkomen.

Ruby's Global Interpreter Lock en Asynchrone Verwerking

Het is belangrijk om de implicaties van Ruby's Global Interpreter Lock (GIL) te begrijpen bij het overwegen van asynchrone thread-gebaseerde verwerking in Ruby.

De GIL is een mechanisme in Ruby's interpreter dat ervoor zorgt dat slechts één thread tegelijk Ruby-code kan uitvoeren, zelfs op multi-core processors. Dit betekent dat hoewel er meerdere threads kunnen worden gemaakt en beheerd binnen een Ruby-proces, slechts één thread actief Ruby-code kan uitvoeren op elk moment.

De GIL is ontworpen om de implementatie van de Ruby-interpreter te vereenvoudigen en thread-veiligheid te bieden voor Ruby's interne datastructuren. Het beperkt echter ook de mogelijkheid voor echte parallelle uitvoering van Ruby-code.

Wanneer je threads gebruikt in Ruby, zoals met de `concurrent-ruby` bibliotheek of de ingebouwde `Thread` klasse, zijn de threads onderworpen aan de beperkingen van de GIL. De GIL staat elke thread toe om Ruby-code uit te voeren voor een korte tijdsperiode voordat er wordt gewisseld naar een andere thread, wat de illusie van gelijktijdige uitvoering creëert.

Echter, door de GIL blijft de daadwerkelijke uitvoering van Ruby-code sequentieel. Terwijl één thread Ruby-code uitvoert, zijn andere threads in feite gepauzeerd, wachtend op hun beurt om de GIL te verkrijgen en uit te voeren.

Dit betekent dat thread-gebaseerde asynchrone verwerking in Ruby het meest

effectief is voor I/O-gebonden taken, zoals wachten op externe API-responses (zoals extern gehoste grote taalmodellen) of het uitvoeren van bestandsI/O-operaties. Wanneer een thread een I/O-operatie tegenkomt, kan deze de GIL vrijgeven, waardoor andere threads kunnen uitvoeren tijdens het wachten op de voltooiing van de I/O.

Aan de andere kant kan de GIL voor CPU-gebonden taken, zoals intensieve berekeningen of langdurige AI-worker verwerking, de potentiële prestatiewinst van thread-gebaseerd parallellisme beperken. Aangezien slechts één thread tegelijk Ruby-code kan uitvoeren, wordt de totale uitvoeringstijd mogelijk niet significant verminderd in vergelijking met sequentiële verwerking.

Om echte parallelle uitvoering voor CPU-gebonden taken in Ruby te bereiken, moet je mogelijk alternatieve benaderingen verkennen, zoals:

- Het gebruik van proces-gebaseerd parallellisme met meerdere Ruby-processen, elk draaiend op een aparte CPU-kern.
- Het benutten van externe bibliotheken of frameworks die native extensies of interfaces bieden naar talen zonder een GIL, zoals C of Rust.,
- Het gebruiken van gedistribueerde computing frameworks of message queues om taken te verdelen over meerdere machines of processen.

Het is cruciaal om de aard van je taken en de beperkingen opgelegd door de GIL te overwegen bij het ontwerpen en implementeren van asynchrone verwerking in Ruby. Hoewel thread-gebaseerde asynchrone verwerking voordelen kan bieden voor I/O-gebonden taken, biedt het mogelijk geen significante prestatieverbeteringen voor CPU-gebonden taken vanwege de beperkingen van de GIL.

Ensembletechnieken voor Verbeterde Nauwkeurigheid

Ensembletechnieken omvatten het combineren van de outputs van meerdere AI-workers om de algemene nauwkeurigheid of robuustheid van het systeem te verbeteren. In plaats van te vertrouwen op een enkele AI-worker, maken ensembletechnieken gebruik van de collectieve intelligentie van meerdere workers om meer geïnformeerde beslissingen te nemen.

Ensembles zijn vooral belangrijk als verschillende delen van je werkstroom het beste werken met verschillende AI-modellen, wat vaker voorkomt dan je misschien denkt. Krachtige modellen zoals GPT-4 zijn extreem duur vergeleken met minder capabele open source alternatieven, en zijn waarschijnlijk niet nodig voor elke afzonderlijke werkstroomstap van je applicatie.

Een veelvoorkomende ensemble-techniek is meerderheidsstemming, waarbij meerdere AI-werkers onafhankelijk van elkaar dezelfde input verwerken, en de uiteindelijke output wordt bepaald door de meerderheid van de consensus. Deze aanpak kan helpen om de impact van individuele werkersfouten te verminderen en de algehele betrouwbaarheid van het systeem te verbeteren.

Laten we een voorbeeld bekijken waarbij we drie AI-werkers hebben voor sentimentanalyse, elk met een verschillend model of voorzien van verschillende contexten. We kunnen hun outputs combineren met behulp van meerderheidsstemming om de uiteindelijke sentimentvoorspelling te bepalen.

```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
```

In dit voorbeeld initialiseert de `SentimentAnalysisEnsemble` klasse met de tekst en roept drie verschillende AI-workers voor sentimentanalyse aan. De `analyze` methode verzamelt de voorspellingen van elke worker en bepaalt het meerderheidssentiment met behulp van de `group_by` en `max_by` methoden. De uiteindelijke uitvoer is het sentiment dat de meeste stemmen krijgt van het ensemble van workers.

Ensembles zijn duidelijk een geval waarbij het experimenteren met parallellisme de moeite waard kan zijn.

Dynamische Selectie en Aanroeping van AI-Workers

In sommige, zo niet de meeste gevallen, kan de specifieke AI-worker die moet worden aangeroepen afhankelijk zijn van runtime-condities of gebruikersinvoer. Dynamische selectie en aanroeping van AI-workers zorgen voor flexibiliteit en aanpasbaarheid in het systeem.

Je zou in de verleiding kunnen komen om veel functionaliteit in één enkele AI-worker te proppen, door deze veel functies te geven en een grote, ingewikkelde prompt die uitlegt hoe je ze moet aanroepen. Weersta deze verleiding, geloof me. Een van de redenen waarom de aanpak die we in dit hoofdstuk bespreken "Multitude of Workers" wordt genoemd, is om ons eraan te herinneren dat het wenselijk is om veel gespecialiseerde workers te hebben, die elk hun eigen kleine taak uitvoeren in dienst van het grotere doel.

Neem bijvoorbeeld een chatbottoepassing waarbij verschillende AI-workers verantwoordelijk zijn voor het afhandelen van verschillende soorten gebruikersvragen. Op basis van de invoer van de gebruiker selecteert de toepassing dynamisch de juiste AI-worker om de vraag te verwerken.

```
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
```

In dit voorbeeld ontvangt de `ChatbotController` een gebruikersvraag via de `process_query` actie. Eerst gebruikt het een `QueryClassifierWorker` om het type

vraag te bepalen. Op basis van het geclassificeerde vraagtype selecteert de controller dynamisch de geschikte AI-worker om het antwoord te genereren. Deze dynamische selectie stelt de chatbot in staat om verschillende soorten vragen te verwerken en ze naar de relevante AI-workers te routeren.

Aangezien het werk van de `QueryClassifierWorker` relatief eenvoudig is en niet veel context of functiedefinities vereist, kun je het waarschijnlijk implementeren met een ultrasnelle kleine LLM zoals `mistralai/mixtral-8x7b-instruct:nitro`. Het heeft mogelijkheden die op veel taken dicht bij GPT-4 niveau komen en, op het moment dat ik dit schrijf, kan Groq het verwerken met een duizelingwekkende doorvoer van 444 tokens per seconde.

Het Combineren van Traditionele NLP met LLMs

Hoewel Grote Taalmodellen (LLMs) het vakgebied van natuurlijke taalverwerking (NLP) hebben gerevolutioneerd en ongeëvenaarde veelzijdigheid en prestaties bieden voor een breed scala aan taken, zijn ze niet altijd de meest efficiënte of kosteneffectieve oplossing voor elk probleem. In veel gevallen kan het combineren van traditionele NLP-technieken met LLMs leiden tot meer geoptimaliseerde, gerichte en economische benaderingen voor het oplossen van specifieke NLP-uitdagingen.

Zie LLMs als de Zwitserse zakmessen van NLP—ongelooflijk veelzijdig en krachtig, maar niet noodzakelijkerwijs het beste gereedschap voor elke klus. Soms kan een specifiek hulpmiddel zoals een kurkentrekker of een blikopener effectiever en efficiënter zijn voor een specifieke taak. Op dezelfde manier kunnen traditionele NLP-technieken, zoals documentclustering, onderwerpsidentificatie en classificatie, vaak meer gerichte en kosteneffectieve oplossingen bieden voor bepaalde aspecten van je NLP-pipeline.

Een van de belangrijkste voordelen van traditionele NLP-technieken is hun computationele efficiëntie. Deze methoden, die vaak vertrouwen op eenvoudigere

statistische modellen of regelgebaseerde benaderingen, kunnen grote hoeveelheden tekstgegevens veel sneller en met minder computationele overhead verwerken in vergelijking met LLMs. Dit maakt ze bijzonder geschikt voor taken die het analyseren en organiseren van grote verzamelingen documenten omvatten, zoals het clusteren van vergelijkbare artikelen of het identificeren van belangrijke onderwerpen binnen een collectie teksten.

Bovendien kunnen traditionele NLP-technieken vaak hoge nauwkeurigheid en precisie bereiken voor specifieke taken, vooral wanneer ze getraind zijn op domeinspecifieke datasets. Een goed afgestelde documentclassificator die gebruik maakt van traditionele machine learning-algoritmen zoals Support Vector Machines (SVM) of Naive Bayes kan bijvoorbeeld documenten nauwkeurig categoriseren in vooraf gedefinieerde categorieën met minimale rekenkosten.

LLMs blinken echter echt uit als het gaat om taken die een dieper begrip van taal, context en redenering vereisen. Hun vermogen om samenhangende en contextueel relevante tekst te genereren, vragen te beantwoorden en lange passages samen te vatten wordt niet geëvenaard door traditionele NLP-methoden. LLMs kunnen effectief omgaan met complexe taalkundige verschijnselen, zoals ambiguïteit, coreferentie en idiomatische uitdrukkingen, waardoor ze onmisbaar zijn voor taken die natuurlijke taalgeneratie of begrip vereisen.

De echte kracht ligt in het combineren van traditionele NLP-technieken met LLMs om hybride benaderingen te creëren die de sterke punten van beide benutten. Door traditionele NLP-methoden te gebruiken voor taken zoals documentvoorverwerking, clustering en onderwerpsextractie, kun je je tekstgegevens efficiënt organiseren en structureren. Deze gestructureerde informatie kan vervolgens worden doorgegeven aan LLMs voor meer geavanceerde taken, zoals het genereren van samenvattingen, het beantwoorden van vragen of het maken van uitgebreide rapporten.

Laten we bijvoorbeeld een gebruikssituatie bekijken waarbij je een trendrapport wilt genereren voor een specifiek domein op basis van een groot corpus van individuele

trenddocumenten. In plaats van uitsluitend te vertrouwen op LLMs, wat computationeel duur en tijdrovend kan zijn voor het verwerken van grote hoeveelheden tekst, kun je een hybride aanpak gebruiken:

1. Gebruik traditionele NLP-technieken, zoals onderwerpsmodellering (bijvoorbeeld Latent Dirichlet Allocation) of clusteringalgoritmen (bijvoorbeeld K-means), om vergelijkbare trenddocumenten te groeperen en belangrijke thema's en onderwerpen binnen het corpus te identificeren.
2. Voer de geclusterde documenten en geïdentificeerde onderwerpen in een LLM, waarbij je gebruik maakt van zijn superieure taalbegreip en generatiemogelijkheden om samenhangende en informatieve samenvattingen te maken voor elk cluster of onderwerp.
3. Gebruik ten slotte het LLM om een uitgebreid trendrapport te genereren door de individuele samenvattingen te combineren, de belangrijkste trends te benadrukken en inzichten en aanbevelingen te geven op basis van de geaggregeerde informatie.

Door traditionele NLP-technieken op deze manier te combineren met LLMs, kun je efficiënt grote hoeveelheden tekstgegevens verwerken, betekenisvolle inzichten extraheren en hoogwaardige rapporten genereren terwijl je de computationele middelen en kosten optimaliseert.

Bij het beginnen van je NLP-projecten is het essentieel om zorgvuldig de specifieke vereisten en beperkingen van elke taak te evalueren en te overwegen hoe traditionele NLP-methoden en LLMs samen kunnen worden ingezet om de beste resultaten te bereiken. Door de efficiëntie en precisie van traditionele technieken te combineren met de veelzijdigheid en kracht van LLMs, kun je zeer effectieve en economische NLP-oplossingen creëren die waarde leveren aan je gebruikers en belanghebbenden.

Gebruik van Tools

In het domein van AI-gedreven applicatieontwikkeling is het concept van “gebruik van tools” of “function calling” naar voren gekomen als een krachtige techniek die je LLM in staat stelt om verbinding te maken met externe tools, APIs, functies, databases en andere bronnen. Deze aanpak maakt een rijkere set aan gedragingen mogelijk dan alleen het uitvoeren van tekst, en zorgt voor meer dynamische interacties tussen je AI-componenten en de rest van het ecosysteem van je applicatie. Zoals we in dit hoofdstuk zullen onderzoeken, geeft het gebruik van tools je ook de mogelijkheid om je AI-model data op gestructureerde manieren te laten genereren.

Wat is Gebruik van Tools?

Gebruik van tools, ook bekend als function calling, is een techniek waarmee ontwikkelaars een lijst van functies kunnen specificeren waarmee een LLM kan

interacteren tijdens het generatieproces. Deze tools kunnen variëren van eenvoudige hulpfuncties tot complexe APIs of database-queries. Door de LLM toegang te geven tot deze tools, kunnen ontwikkelaars de mogelijkheden van het model uitbreiden en het in staat stellen taken uit te voeren die externe kennis of acties vereisen.

Figuur 8. Voorbeeld van een functiedefinitie voor een AI-medewerker die documenten analyseert

```
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
```

Het kernidee achter het gebruik van tools is om het LLM de mogelijkheid te geven om dynamisch de juiste tools te selecteren en uit te voeren op basis van de input van de gebruiker of de taak die voorhanden is. In plaats van uitsluitend te vertrouwen op de vooraf getrainde kennis van het model, stelt het gebruik van tools het LLM in staat

om externe bronnen te benutten voor het genereren van nauwkeurigere, relevantere en bruikbare antwoorden. Het gebruik van tools maakt technieken zoals RAG (Retrieval Augmented Generation) veel eenvoudiger te implementeren dan anders het geval zou zijn.

> Merk op dat, tenzij anders vermeld, dit boek ervan uitgaat dat uw AI-model geen toegang heeft tot ingebouwde server-side tools. Alle tools die u beschikbaar wilt maken voor uw AI moeten expliciet door u worden gedeclareerd in elke API-aanvraag, met voorzieningen voor de uitvoering ervan wanneer uw AI aangeeft dat het die tool in zijn antwoord wil gebruiken.

De Potentie van Tool Gebruik

Het gebruik van tools opent een breed scala aan mogelijkheden voor AI-gedreven toepassingen. Hier zijn enkele voorbeelden van wat er bereikt kan worden met het gebruik van tools:

1. **Chatbots en Virtuele Assistenten**: Door een LLM te verbinden met externe tools kunnen chatbots en virtuele assistenten complexere taken uitvoeren, zoals het ophalen van informatie uit databases, het uitvoeren van API-aanroepen of het interacteren met andere systemen. Een chatbot zou bijvoorbeeld een CRM-tool kunnen gebruiken om de status van een deal te wijzigen op basis van het verzoek van de gebruiker.
2. **Data-analyse en Inzichten**: LLM's kunnen worden verbonden met data-analysetools of bibliotheken om geavanceerde gegevensverwerkingstaken uit te voeren. Dit maakt het mogelijk voor applicaties om inzichten te genereren, vergelijkende analyses uit te voeren of datagestuurde aanbevelingen te doen op basis van gebruikersvragen.

3. **Zoeken en Informatieophaling**: Het gebruik van tools stelt LLM's in staat om te interacteren met zoekmachines, vectordatabases of andere informatieophalingssystemen. Door gebruikersvragen om te zetten in zoekopdrachten kan het LLM relevante informatie uit meerdere bronnen ophalen en uitgebreide antwoorden geven op vragen van gebruikers.
4. **Integratie met Externe Diensten**: Het gebruik van tools maakt naadloze integratie mogelijk tussen AI-gedreven applicaties en externe diensten of API's. Een LLM zou bijvoorbeeld kunnen interacteren met een weer-API om realtime weerupdates te geven of met een vertaal-API om meertalige antwoorden te genereren.

Het Tool Gebruik Werkproces

Het werkproces voor het gebruik van tools bestaat meestal uit vier belangrijke stappen:

1. Functiedefinities opnemen in je request context
2. Dynamische (of expliciete) toolselectie
3. Uitvoering van functie(s)
4. Optionele voortzetting van de oorspronkelijke prompt

Laten we elk van deze stappen in detail bekijken.

Functiedefinities opnemen in je request context

De AI weet welke tools het tot zijn beschikking heeft omdat je een lijst meegeeft als onderdeel van je completion request (meestal gedefinieerd als functies met behulp van een variant van JSON schema).

De precieze syntax van tooldefinitie is modelspecifiek.

Dit is hoe je een `get_weather` functie definieert in 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"]
    }
}
```

En zo zou je dezelfde functie voor GPT-4 definiëren, waarbij je deze als waarde voor de `tools` parameter doorgeeft:

```
{
    "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"],
```

```
    },
}
```

> Bijna hetzelfde, behalve anders zonder duidelijke reden! Wat vervelend.

Functiedefinities specificeren naam, beschrijving en invoerparameters. Invoerparameters kunnen verder worden gedefinieerd met behulp van attributen zoals enums om de acceptabele waarden te beperken, en door aan te geven of een parameter vereist is of niet.

Naast de eigenlijke functiedefinities kun je ook instructies of context opnemen over waarom en hoe je de functie moet gebruiken in de systeemrichtlijn.

Bijvoorbeeld, mijn Webzoekfunctie in Olympia bevat deze systeemrichtlijn, die de AI eraan herinnert dat het de genoemde hulpmiddelen tot zijn beschikking heeft:

```
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.
```

Het geven van gedetailleerde beschrijvingen wordt beschouwd als de belangrijkste factor in de prestaties van hulpmiddelen. Je beschrijvingen moeten elk detail over het hulpmiddel uitleggen, waaronder:

- Wat het hulpmiddel doet
- Wanneer het gebruikt moet worden (en wanneer niet)
- Wat elke parameter betekent en hoe deze het gedrag van het hulpmiddel beïnvloedt

- Alle belangrijke kanttekeningen of beperkingen die van toepassing zijn op de implementatie van het hulpmiddel

Hoe meer context je de AI kunt geven over je hulpmiddelen, hoe beter deze zal zijn in het beslissen wanneer en hoe ze te gebruiken. Zo beveelt Anthropic voor zijn Claude 3-serie minimaal 3-4 zinnen per hulpmiddelbeschrijving aan, meer als het hulpmiddel complex is.

Het is niet per se intuïtief, maar beschrijvingen worden ook als belangrijker beschouwd dan voorbeelden. Hoewel je voorbeelden van het gebruik van een hulpmiddel kunt opnemen in de beschrijving of in de bijbehorende prompt, is dit minder belangrijk dan een duidelijke en uitgebreide uitleg van het doel en de parameters van het hulpmiddel. Voeg pas voorbeelden toe nadat je de beschrijving volledig hebt uitgewerkt.

Hier is een voorbeeld van een Stripe-achtige API-functiespecificatie:

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

In de praktijk hebben sommige modellen moeite met het omgaan met geneste functiespecificaties en complexe outputgegevenstypen zoals arrays, dictionaries etc. Maar in theorie zou je JSON Schema-specificaties van willekeurige diepte moeten kunnen aanleveren!

Dynamische Gereedschapsselectie

Wanneer je een chatvervolmaking uitvoert die gereedschapsdefinities bevat, selecteert het LLM dynamisch de meest geschikte gereedschappen om te gebruiken en genereert het de vereiste invoerparameters voor elk gereedschap.

In de praktijk is het vermogen van de AI om *precies* de juiste functie aan te roepen en *exact* je specificatie voor de invoer te volgen wisselvallig. Het verlagen van de temperatuur-hyperparameter naar `0.0` helpt aanzienlijk, maar uit ervaring krijg je nog steeds af en toe fouten. Deze fouten omvatten gehallucineerde functienamen, verkeerd benoemde of simpelweg ontbrekende invoerparameters. Parameters worden doorgegeven als JSON, wat betekent dat je soms fouten ziet veroorzaakt door afgekapte, verkeerd geciteerde of anderszins beschadigde JSON.

Zelfherstellende Data-patronen kunnen helpen bij het automatisch repareren van functieaanroepen die mislukken door syntaxisfouten.

Geforceerde (oftewel Expliciete) Gereedschapsselectie

Sommige modellen bieden de mogelijkheid om het aanroepen van een bepaalde functie te forceren, als parameter in het verzoek. Anders is de beslissing om de functie al dan niet aan te roepen volledig aan het oordeel van de AI.

Het vermogen om een functieaanroep te forceren is cruciaal in bepaalde scenario's waar je wilt verzekeren dat een specifiek gereedschap of functie wordt uitgevoerd, ongeacht het dynamische selectieproces van de AI. Er zijn verschillende redenen waarom deze mogelijkheid belangrijk is:

1. **Expliciete Controle**: Je gebruikt de AI mogelijk als een *Discrete Component* of in een voorgedefinieerde workflow die de uitvoering van een bepaalde functie op een bepaald moment vereist. Door de aanroep te forceren, kun je garanderen dat

de gewenste functie wordt aangeroepen in plaats van de AI vriendelijk te moeten vragen het te doen.

2. **Debuggen en Testen**: Bij het ontwikkelen en testen van AI-gestuurde applicaties is het vermogen om functieaanroepen te forceren van onschatbare waarde voor debugdoeleinden. Door expliciet specifieke functies te activeren, kun je individuele componenten van je applicatie isoleren en testen. Dit stelt je in staat om de juistheid van de functie-implementaties te verifiëren, de invoerparameters te valideren en te verzekeren dat de verwachte resultaten worden teruggegeven.
3. **Omgaan met Randgevallen**: Er kunnen randgevallen of uitzonderlijke scenario's zijn waarbij het dynamische selectieproces van de AI ervoor zou kunnen kiezen om een functie niet uit te voeren terwijl dat wel zou moeten, en je weet dat op basis van externe processen. In dergelijke gevallen stelt het vermogen om een functieaanroep te forceren je in staat om deze situaties expliciet af te handelen. Definieer regels of voorwaarden in je applicatielogica om te bepalen wanneer je het oordeel van de AI moet overschrijven.
4. **Consistentie en Reproduceerbaarheid**: Als je een specifieke reeks functies hebt die in een bepaalde volgorde moet worden uitgevoerd, garandeert het forceren van de aanroepen dat dezelfde volgorde elke keer wordt gevolgd. Dit is vooral belangrijk in applicaties waar consistentie en voorspelbaar gedrag kritiek zijn, zoals in financiële systemen of wetenschappelijke simulaties.
5. **Prestatie-optimalisatie**: In sommige gevallen kan het forceren van een functieaanroep leiden tot prestatie-optimalisaties. Als je weet dat een specifieke functie vereist is voor een bepaalde taak en dat het dynamische selectieproces van de AI onnodige overhead zou kunnen introduceren, kun je het selectieproces omzeilen en direct de vereiste functie aanroepen. Dit kan helpen om de latentie te verminderen en de algehele efficiëntie van je applicatie te verbeteren.

Samenvattend biedt het vermogen om functieaanroepen te forceren in AI-gestuurde applicaties expliciete controle, helpt bij het debuggen en testen, handelt randgevallen

af en verzekert consistentie en reproduceerbaarheid. Het is een krachtig gereedschap in je arsenaal, maar we moeten nog één aspect van deze belangrijke functie bespreken.

In veel besluitvormingstoepassingen willen we altijd dat het model een functieaanroep doet en willen we mogelijk nooit dat het model alleen met zijn interne kennis reageert. Als je bijvoorbeeld routeert tussen meerdere modellen die gespecialiseerd zijn in verschillende taken (meertalige invoer, wiskunde, etc.), kun je het functie-aanroepende model gebruiken om verzoeken te delegeren naar een van de hulpmodellen en nooit zelfstandig te laten reageren.

Tool Choice Parameter

GPT-4 en andere taalmodellen die functieaanroepen ondersteunen, geven je een `tool_choice` parameter voor het controleren of gereedschapsgebruik vereist is als onderdeel van een vervolmaking. Deze parameter heeft drie mogelijke waarden:

- `auto` geeft de AI volledige vrijheid over het gebruik van een gereedschap of simpelweg reageren
- `required` vertelt de AI dat het een gereedschap *moet* aanroepen *in plaats van* te reageren, maar laat de selectie van het gereedschap over aan de AI
- De derde optie is het instellen van de parameter van de `name_of_function` die je wilt forceren. Meer daarover in de volgende sectie.

Merk op dat als je tool choice op `required` zet, het model gedwongen wordt om de meest relevante functie te kiezen uit de beschikbare functies, zelfs als geen enkele echt bij de prompt past. Op het moment van publicatie ken ik geen model dat een lege `tool_calls` response zal teruggeven, of op een andere manier laat weten dat het geen geschikte functie heeft gevonden om aan te roepen.

Een Functie Forceren voor Gestructureerde Uitvoer

De mogelijkheid om een functie-aanroep te forceren geeft je een manier om gestructureerde data uit een chatvoltooing te krijgen in plaats van het zelf uit het platte tekst-antwoord te moeten extraheren.

> Waarom is het forceren van functies voor gestructureerde uitvoer zo belangrijk? Simpel gezegd, omdat het extraheren van gestructureerde data uit LLM-uitvoer een hele klus is. Je kunt het jezelf wat makkelijker maken door om data in XML te vragen, maar dan moet je nog steeds XML verwerken. En wat doe je als die XML ontbreekt omdat je AI antwoordde: "Het spijt me, maar ik kan de gevraagde gegevens niet genereren omdat bla, bla, bla..."

Bij het gebruik van tools op deze manier:

- Zou je waarschijnlijk één tool in je verzoek moeten definiëren
- Vergeet niet het gebruik van de functie te forceren met de `tool_choice` parameter
- Onthoud dat het model de input naar de tool stuurt, dus de naam van de tool en beschrijving moeten vanuit het perspectief van het model zijn, niet dat van jou

Dit laatste punt verdient een voorbeeld ter verduidelijking. Stel dat je de AI vraagt om sentimentanalyse uit te voeren op gebruikerstekst. De naam van de functie zou dan niet `analyze_sentiment` zijn, maar eerder iets als `save_sentiment_analysis`. De AI is degene die de sentimentanalyse uitvoert, *niet de tool*. Het enige wat de tool doet (vanuit het perspectief van de AI) is het opslaan van de resultaten van de analyse.

Hier is een voorbeeld van het gebruik van Claude 3 om een samenvatting van een afbeelding vast te leggen in goed gestructureerde JSON, deze keer vanaf de opdrachtregel met behulp van `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."
                }
            ]
        }
    ]
}'
```

In het gegeven voorbeeld gebruiken we het Claude 3-model van Anthropic om een gestructureerde JSON-samenvatting van een afbeelding te genereren. Zo werkt het:

1. We definiëren een enkele tool genaamd `record_summary` in de `tools`-array van de request payload. Deze tool is verantwoordelijk voor het vastleggen van een samenvatting van de afbeelding in goed gestructureerde JSON.
2. De `record_summary`-tool heeft een `input_schema` dat de verwachte structuur van de JSON-uitvoer specificeert. Het definieert drie eigenschappen:

    - `key_colors`: Een array van objecten die de belangrijkste kleuren in de afbeelding vertegenwoordigen. Elk kleurobject heeft eigenschappen voor de rood-, groen- en blauwwaarden (variërend van 0.0 tot 1.0) en een menselijk leesbare kleurnaam in snake_case-formaat.
    - `description`: Een string-eigenschap voor een korte beschrijving van de afbeelding, beperkt tot 1-2 zinnen.
    - `estimated_year`: Een optionele integer-eigenschap voor het geschatte jaar waarin de foto is genomen, als het een niet-fictieve foto lijkt te zijn.

3. In de `messages`-array leveren we de afbeeldingsgegevens aan als een base64-gecodeerde string samen met het mediatype. Hierdoor kan het model de afbeelding verwerken als onderdeel van de invoer.
4. We geven Claude ook de opdracht om de `record_summary`-tool te gebruiken om de afbeelding te beschrijven.
5. Wanneer het verzoek naar het Claude 3-model wordt gestuurd, analyseert het de afbeelding en genereert het een JSON-samenvatting op basis van het gespecificeerde `input_schema`. Het model extraheert de belangrijkste kleuren, geeft een korte beschrijving en schat het jaar waarin de afbeelding is genomen (indien van toepassing).
6. De gegenereerde JSON-samenvatting wordt doorgegeven als parameters aan de `record_summary`-tool, wat zorgt voor een gestructureerde weergave van de belangrijkste kenmerken van de afbeelding.

Door de `record_summary`-tool te gebruiken met een goed gedefinieerd `input_schema`, kunnen we een gestructureerde JSON-samenvatting van een

afbeelding verkrijgen zonder te vertrouwen op extractie van platte tekst. Deze aanpak zorgt ervoor dat de uitvoer een consistent formaat volgt en gemakkelijk kan worden geparseerd en verwerkt door downstream componenten van de applicatie.

De mogelijkheid om een functieaanroep af te dwingen en de verwachte uitvoerstructuur te specificeren is een krachtige functie van toolgebruik in AI-gestuurde applicaties. Het stelt ontwikkelaars in staat meer controle te hebben over de gegenereerde uitvoer en vereenvoudigt de integratie van door AI gegenereerde gegevens in de workflow van hun applicatie.

Uitvoering van Functie(s)

Je hebt functies gedefinieerd en je AI geprompt, die heeft besloten dat het een van je functies moet aanroepen. Nu is het tijd voor je applicatiecode of bibliotheek, als je een Ruby gem zoals `raix-rails` gebruikt, om de functieaanroep en zijn parameters naar de corresponderende implementatie *in je applicatiecode* te dispatchen.

Je applicatiecode bepaalt wat er met de resultaten van de functie-uitvoering moet gebeuren. Misschien betreft dit een enkele regel code in een lambda, of misschien betreft het het aanroepen van een externe API. Misschien betreft het het aanroepen van een andere AI-component, of misschien betreft het honderden of zelfs duizenden regels code in de rest van je systeem. Het is geheel aan jou.

Soms is de functieaanroep het einde van de operatie, maar als de resultaten informatie vertegenwoordigen in een ketening van gedachten die door de AI moet worden voortgezet, dan moet je applicatiecode de uitvoeringsresultaten in het chattranscript invoegen en de AI laten doorgaan met verwerken.

Hier bijvoorbeeld een Raix-functiedeclaratie die wordt gebruikt door Olympia's `AccountManager` om te communiceren met onze klanten als onderdeel van een Intelligente Werkstroomorganisatie voor klantenservice.

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

  # lots of other functions...

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

Het is misschien niet direct duidelijk wat hier gebeurt, dus ik zal het uitleggen.

1. De `AccountManager` klasse definieert veel functies die gerelateerd zijn aan accountbeheer. Het kan je abonnement wijzigen, teamleden toevoegen en verwijderen, en nog veel meer.
2. De instructies op het hoogste niveau vertellen `AccountManager` dat het de accounteigenaar moet informeren over de resultaten van het accountwijzigingsverzoek, met behulp van de `notify_account_owner` functie.
3. De beknopte definitie van de functie bevat:

- naam
- beschrijving
- parameters `message: { type: "string" }`
- een blok code dat wordt uitgevoerd wanneer de functie wordt aangeroepen

Na het bijwerken van het transcript met de resultaten van het functieblok, wordt de `chat_completion` methode opnieuw aangeroepen. Deze methode is verantwoordelijk

voor het terugsturen van het bijgewerkte gespreksverloop naar het AI-model voor verdere verwerking. We noemen dit proces een *conversatielus*.

Wanneer het AI-model een nieuw chat completion verzoek ontvangt met een bijgewerkt transcript, heeft het toegang tot de resultaten van de eerder uitgevoerde functie. Het kan deze resultaten analyseren, ze meenemen in zijn besluitvormingsproces en de volgende respons of actie genereren op basis van de cumulatieve context van het gesprek. Het kan ervoor kiezen om extra functies uit te voeren op basis van de bijgewerkte context, of het kan een definitieve respons genereren op de oorspronkelijke prompt als het bepaalt dat er geen verdere functieaanroepen nodig zijn.

Optionele Voortzetting van de Oorspronkelijke Prompt

Wanneer je de gereedschapsresultaten terugstuurt naar de LLM en doorgaat met de verwerking van de oorspronkelijke prompt, gebruikt de AI deze resultaten om ofwel aanvullende functies aan te roepen ofwel een definitieve platte tekst respons te genereren.

Sommige modellen zoals Cohere's Command-R kunnen specifiek vermelden welke gereedschappen ze hebben gebruikt in hun antwoorden, wat zorgt voor extra transparantie en traceerbaarheid.

Afhankelijk van het gebruikte model zullen de resultaten van de functieaanroep zich bevinden in transcriptberichten met hun eigen speciale rol of worden weergegeven in een andere syntax. Maar het belangrijkste is dat die gegevens in het transcript staan, zodat de AI ze kan meenemen in zijn beslissing over wat er vervolgens moet gebeuren.

Een veelvoorkomende (en potentieel dure) fout is het vergeten om de functieresultaten aan het transcript toe te voegen voordat je verdergaat met het gesprek. Hierdoor zal de AI op vrijwel dezelfde manier worden aangeroepen als voordat deze de functie voor de eerste keer aanriep. Met andere woorden, voor zover de AI weet, heeft deze de functie nog niet aangeroepen. Dus roept hij deze opnieuw aan. En opnieuw. En opnieuw, tot in het oneindige totdat je het onderbreekt. Hopelijk was je context niet te groot en je model niet te duur!

Best Practices voor Gereedschapsgebruik

Om het meeste uit gereedschapsgebruik te halen, kun je de volgende best practices overwegen.

Beschrijvende Definities

Zorg voor duidelijke en beschrijvende namen en beschrijvingen voor elk gereedschap en zijn invoerparameters. Dit helpt de LLM beter te begrijpen wat het doel en de mogelijkheden van elk gereedschap zijn.

> Ik kan je uit ervaring vertellen dat de algemene wijsheid die zegt dat “naamgeving moeilijk is” hier ook geldt; ik heb dramatisch verschillende resultaten gezien van LLMs alleen door het veranderen van functienamen of de formulering van beschrijvingen. Soms verbetert het verwijderen van beschrijvingen de prestaties zelfs.

Verwerking van Gereedschapsresultaten

Bij het doorgeven van gereedschapsresultaten aan de LLM, zorg ervoor dat ze goed gestructureerd en volledig zijn. Gebruik betekenisvolle sleutels en waarden om de output van elk gereedschap weer te geven. Experimenteer met verschillende formaten en kijk welke het beste werkt, van JSON tot platte tekst.

De Resultaatinterpreter pakt deze uitdaging aan door AI te gebruiken om de resultaten te analyseren en mensvriendelijke uitleg, samenvattingen of belangrijke inzichten te geven.

Foutafhandeling

Implementeer robuuste foutafhandelingsmechanismen om gevallen af te handelen waarbij de LLM mogelijk ongeldige of niet-ondersteunde invoerparameters voor gereedschapsaanroepen genereert. Handel fouten die kunnen optreden tijdens de uitvoering van gereedschap netjes af en herstel ervan.

Een bijzonder prettige eigenschap van de AI is dat het foutmeldingen begrijpt! Dit betekent dat als je in een snelle en praktische mindset werkt, je simpelweg alle uitzonderingen kunt opvangen die gegenereerd worden in de implementatie van een gereedschap, en deze kunt terugsturen naar de AI zodat het weet wat er is gebeurd!

Hier is bijvoorbeeld een afgeslankte versie van de implementatie van Google zoeken in 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
```

Google-zoekopdrachten in Olympia zijn een proces in twee stappen. Eerst voer je de zoekopdracht uit, daarna vat je de resultaten samen. Als er een fout optreedt, ongeacht welke, wordt de foutmelding verpakt en teruggestuurd naar de AI. Deze techniek vormt de basis van vrijwel alle *Intelligente Foutafhandeling*-patronen.

Stel bijvoorbeeld dat de `GoogleSearch` API-aanroep mislukt vanwege een 503 Service Unavailable-foutmelding. Die wordt doorgegeven naar de hoogste rescue-clausule, en de beschrijving van de fout wordt als resultaat van de functieaanroep teruggestuurd naar de AI. In plaats van de gebruiker een leeg scherm of technische fout te tonen, zegt de AI iets als "Het spijt me, maar ik heb momenteel geen toegang tot mijn Google-zoekfuncties. Ik kan het later opnieuw proberen als u dat wilt."

Dit lijkt misschien slechts een slimme truc, maar denk eens aan een ander soort fout, waarbij de AI een externe API aanriep en directe controle had over de parameters die aan de API moesten worden doorgegeven. Misschien maakte hij een fout in hoe hij die parameters genereerde? Mits de foutmelding van de externe API gedetailleerd genoeg is, betekent het terugsturen van de foutmelding naar de aanroepende AI dat deze de parameters kan heroverwegen en het opnieuw kan proberen. Automatisch. Ongeacht wat de fout was.

Bedenk nu eens wat er nodig zou zijn om dat soort robuuste foutafhandeling in *normale* code te repliceren. Het is praktisch onmogelijk.

Iteratieve Verfijning

Als het LLM niet de juiste hulpmiddelen aanbeveelt of suboptimale responses genereert, itereer dan op de gereedschapsdefinities, beschrijvingen en invoerparameters. Verfijn en verbeter de hulpmiddelenopzet voortdurend op basis van het waargenomen gedrag en de gewenste uitkomsten.

1. Begin met eenvoudige gereedschapsdefinities: Start met het definiëren van hulpmiddelen met duidelijke en beknopte namen, beschrijvingen en invoerparameters. Vermijd aanvankelijk een te ingewikkelde opzet van hulpmiddelen en concentreer je op de kernfunctionaliteit. Als je bijvoorbeeld de resultaten van sentimentanalyse wilt opslaan, begin dan met een basisdefinitie zoals:

```
{
  "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. Test en observeer: Zodra je de eerste gereedschapsdefinities hebt opgezet, test je ze met verschillende prompts en observeer je hoe het GTM met het gereedschap interacteert. Let op de kwaliteit en relevantie van de gegenereerde responses. Als het GTM suboptimale responses genereert, is het tijd om de gereedschapsdefinities te verfijnen.

3. Verfijn beschrijvingen: Als het GTM het doel van een tool verkeerd begrijpt, probeer dan de beschrijving van het gereedschap te verfijnen. Voorzie het van meer context, voorbeelden of verduidelijkingen om het GTM te begeleiden bij het effectief gebruik van het gereedschap. Je kunt bijvoorbeeld de beschrijving van het sentimentanalyse-gereedschap bijwerken om specifieker in te gaan op de *emotionele toon* van de te analyseren tekst:

```
{
  "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. Pas invoerparameters aan: Als de LLM ongeldige of irrelevante invoerparameters voor een tool genereert, overweeg dan om de parameterdefinities aan te passen. Voeg specifiekere beperkingen, validatieregels of voorbeelden toe om het verwachte invoerformaat te verduidelijken.
5. Itereer op basis van feedback: Monitor voortdurend de prestaties van je tools en verzamel feedback van gebruikers of belanghebbenden. Gebruik deze feedback om verbeterpunten te identificeren en breng iteratieve verfijningen aan in de gereedschapsdefinities. Als gebruikers bijvoorbeeld melden dat de analyse niet goed omgaat met sarcasme, kun je een opmerking toevoegen in de beschrijving:

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

Door je gereedschapsdefinities iteratief te verfijnen op basis van geobserveerd gedrag en feedback, kun je de prestaties en effectiviteit van je AI-gestuurde applicatie geleidelijk verbeteren. Onthoud dat je de gereedschapsdefinities helder, bondig en gefocust moet houden op de specifieke taak. Test en valideer de gereedschapsinteracties regelmatig om ervoor te zorgen dat ze in lijn zijn met je gewenste resultaten.

Samenstellen en Aaneenschakelen van Gereedschappen

Een van de krachtigste aspecten van gereedschapsgebruik, waar tot nu toe alleen op gezinspeeld is, is de mogelijkheid om meerdere gereedschappen samen te stellen en aan elkaar te koppelen om complexe taken uit te voeren. Door je gereedschapsdefinities en hun invoer-/uitvoerformaten zorgvuldig te ontwerpen, kun je herbruikbare bouwstenen creëren die op verschillende manieren kunnen worden gecombineerd.

Laten we een voorbeeld bekijken waarbij je een gegevensanalysepijplijn bouwt voor je AI-gestuurde applicatie. Je zou de volgende gereedschappen kunnen hebben:

1. **`DataRetrieval`**: Een gereedschap dat gegevens ophaalt uit een database of API op basis van specifieke criteria.
2. **`DataProcessing`**: Een gereedschap dat berekeningen, transformaties of aggregaties uitvoert op de opgehaalde gegevens.
3. **`DataVisualization`**: Een gereedschap dat de verwerkte gegevens presenteert in een gebruiksvriendelijk formaat, zoals grafieken of diagrammen.

Door deze gereedschappen aan elkaar te koppelen, kun je een krachtige werkstroom creëren die relevante gegevens ophaalt, verwerkt en de resultaten op een betekenisvolle manier presenteert. Zo zou de gereedschapsworkflow eruit kunnen zien:

1. De LLM ontvangt een gebruikersvraag met het verzoek om inzichten in verkoopgegevens voor een specifieke productcategorie.
2. De LLM selecteert het `DataRetrieval`-gereedschap en genereert de juiste invoerparameters om de relevante verkoopgegevens uit de database op te halen.
3. De opgehaalde gegevens worden "doorgegeven" aan het `DataProcessing`-gereedschap, dat metrics berekent zoals totale omzet, gemiddelde verkoopprijs en groeipercentage.
4. De verwerkte gegevens worden vervolgens verwerkt door het `DataVisualization`-gereedschap, dat een visueel aantrekkelijke grafiek of diagram maakt om de inzichten weer te geven, waarbij de URL van de grafiek wordt teruggegeven aan de LLM.
5. Tot slot genereert de LLM een geformatteerd antwoord op de gebruikersvraag met behulp van markdown, waarbij de gevisualiseerde gegevens worden geïntegreerd en een samenvatting van de belangrijkste bevindingen wordt gegeven.

Door deze gereedschappen samen te stellen, kun je een naadloze gegevensanalyseworkflow creëren die eenvoudig in je applicatie kan worden geïntegreerd. Het mooie van deze aanpak is dat elk gereedschap onafhankelijk kan worden ontwikkeld en getest, en vervolgens op verschillende manieren kan worden gecombineerd om diverse problemen op te lossen.

Om een soepele samenstelling en aaneenschakeling van gereedschappen mogelijk te maken, is het belangrijk om duidelijke invoer- en uitvoerformaten voor elk gereedschap te definiëren.

Het `DataRetrieval`-gereedschap zou bijvoorbeeld parameters kunnen accepteren zoals de databaseverbindingsgegevens, tabelnaam en queryvoorwaarden, en

het resultaat kunnen retourneren als een gestructureerd JSON-object. Het `DataProcessing`-gereedschap kan dan dit JSON-object als invoer verwachten en een getransformeerd JSON-object als uitvoer produceren. Door de gegevensstroom tussen gereedschappen te standaardiseren, kun je compatibiliteit en herbruikbaarheid waarborgen.

Denk bij het ontwerpen van je gereedschapsecosysteem na over hoe verschillende gereedschappen kunnen worden gecombineerd om veelvoorkomende gebruikssituaties in je applicatie aan te pakken. Overweeg om gereedschappen op hoog niveau te creëren die veelvoorkomende workflows of bedrijfslogica omvatten, waardoor het voor de LLM gemakkelijker wordt om ze effectief te selecteren en te gebruiken.

Onthoud dat de kracht van gereedschapsgebruik ligt in de flexibiliteit en modulariteit die het biedt. Door complexe taken op te delen in kleinere, herbruikbare gereedschappen, kun je een robuuste en aanpasbare AI-gestuurde applicatie creëren die een breed scala aan uitdagingen kan aanpakken.

Toekomstige Ontwikkelingen

Naarmate het gebied van AI-gestuurde applicatieontwikkeling evolueert, kunnen we verdere vooruitgang in gereedschapsgebruiksmogelijkheden verwachten. Enkele potentiële toekomstige richtingen zijn:

1. **Meerstaps-gereedschapsgebruik**: LLM's kunnen mogelijk bepalen hoe vaak ze gereedschappen moeten gebruiken om een bevredigend antwoord te genereren. Dit kan meerdere rondes van gereedschapsselectie en -uitvoering omvatten op basis van tussenresultaten.
2. **Voorgedefinieerde Gereedschappen**: AI-platforms kunnen mogelijk een set voorgedefinieerde gereedschappen aanbieden die ontwikkelaars direct kunnen gebruiken, zoals Python-interpreters, zoekgereedschappen voor het web of algemene hulpfuncties.

3. **Naadloze Integratie**: Naarmate gereedschapsgebruik gangbaarder wordt, kunnen we betere integratie verwachten tussen AI-platforms en populaire ontwikkelingsframeworks, waardoor het voor ontwikkelaars gemakkelijker wordt om gereedschapsgebruik in hun applicaties te integreren.

Gereedschapsgebruik is een krachtige techniek die ontwikkelaars in staat stelt om het volledige potentieel van LLM's in AI-gestuurde applicaties te benutten. Door LLM's te verbinden met externe gereedschappen en bronnen, kun je meer dynamische, intelligente en contextbewuste systemen creëren die zich kunnen aanpassen aan gebruikersbehoeften en waardevolle inzichten en acties kunnen bieden.

Hoewel gereedschapsgebruik enorme mogelijkheden biedt, is het belangrijk om je bewust te zijn van potentiële uitdagingen en overwegingen. Een belangrijk aspect is het beheren van de complexiteit van gereedschapsinteracties en het waarborgen van de stabiliteit en betrouwbaarheid van het gehele systeem. Je moet scenario's afhandelen waarbij gereedschapsaanroepen kunnen mislukken, onverwachte resultaten kunnen opleveren of prestatie-implicaties kunnen hebben. Daarnaast moet je beveiligings- en toegangscontrolemaatregelen overwegen om ongeautoriseerd of kwaadwillig gebruik van gereedschappen te voorkomen. Goede foutafhandeling, logging en monitoringmechanismen zijn cruciaal om de integriteit en prestaties van je AI-gestuurde applicatie te behouden.

Tijdens het verkennen van de mogelijkheden van gereedschapsgebruik in je eigen projecten, is het belangrijk om te beginnen met duidelijke doelstellingen, goed gestructureerde gereedschapsdefinities, en te itereren op basis van feedback en resultaten. Met de juiste aanpak en mindset kan gereedschapsgebruik nieuwe niveaus van innovatie en waarde ontsluiten in je AI-gestuurde toepassingen

Streamverwerking

Het streamen van data over HTTP, ook bekend als server-sent events (SSE), is een mechanisme waarbij de server continu data naar de client stuurt zodra deze beschikbaar komt, zonder dat de client hier expliciet om hoeft te vragen. Aangezien de reactie van de AI stapsgewijs wordt gegenereerd, is het logisch om een responsieve gebruikerservaring te bieden door de uitvoer van de AI weer te geven terwijl deze wordt gegenereerd. En eigenlijk bieden alle AI-provider API's die ik ken streaming-responses als optie in hun completion endpoints.

De reden dat dit hoofdstuk hier in het boek verschijnt, direct na Using Tools, is vanwege hoe krachtig het kan zijn om het gebruik van tools te combineren met live AI-responses aan gebruikers. Dit maakt dynamische en interactieve ervaringen mogelijk waarbij de AI gebruikersinvoer kan verwerken, verschillende tools en functies naar eigen inzicht kan gebruiken en vervolgens realtime responses kan geven.

Om deze naadloze interactie te bereiken, moet je streamverwerkers schrijven die zowel door AI aangeroepen toolfuncties als platte tekstuitvoer naar de eindgebruiker kunnen versturen. De noodzaak om te loopen na het verwerken van een toolfunctie voegt een interessante uitdaging toe aan de taak.

Implementatie van een ReplyStream

Om te demonstreren hoe streamverwerking kan worden geïmplementeerd, zal dit hoofdstuk diep ingaan op een vereenvoudigde versie van de `ReplyStream`-klasse die wordt gebruikt in Olympia. Instanties van deze klasse kunnen worden doorgegeven als de `stream`-parameter in AI-clientbibliotheken zoals ruby-openai en openrouter

Hier is hoe ik `ReplyStream` gebruik in Olympia's `PromptSubscriber`, die via Wisper luistert naar de creatie van nieuwe gebruikersberichten.

```
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...
```

Naast een `context`-referentie naar de prompt-abonnee die het heeft geïnstantieerd, heeft de `ReplyStream`-klasse ook instantievariabelen om een buffer van ontvangen gegevens op te slaan, en arrays om functienamen en argumenten bij te houden die tijdens de streamverwerking worden aangeroepen.

```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
```

De `initialize` methode zet de beginstatus van de `ReplyStream` instantie op, waarbij de buffer, context en andere variabelen worden geïnitialiseerd.

De `call` methode is het belangrijkste toegangspunt voor het verwerken van de streaminggegevens. Deze methode accepteert een `chunk` aan gegevens (weergegeven als een hash) en een optionele `bytesize` parameter, die in ons voorbeeld niet wordt gebruikt. Binnen deze methode gebruikt de klasse patroonherkenning om verschillende scenario's af te handelen op basis van de structuur van de ontvangen chunk.

Het aanroepen van `deep_symbolize_keys` op de chunk maakt de patroonherkenning eleganter, doordat we kunnen werken met symbolen in plaats van strings.

```
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
```

Het eerste patroon waar we naar zoeken is een gereedschapsaanroep samen met de bijbehorende functienaam. Als we er een detecteren, plaatsen we deze in de `f_name` array. We slaan functienamen op in een geïndexeerde array, omdat het model in staat is tot parallelle functieaanroepen, waarbij meerdere functies tegelijk ter uitvoering worden verzonden.

> Parallelle functieaanroepen is het vermogen van een AI-model om meerdere functieaanroepen tegelijk uit te voeren, waarbij de effecten en resultaten van deze functieaanroepen parallel kunnen worden verwerkt. Dit is vooral nuttig als functies veel tijd in beslag nemen, en het vermindert het aantal communicatierondes met de API, wat op zijn beurt een aanzienlijke besparing in tokenverbruik kan opleveren.

Vervolgens moeten we zoeken naar de argumenten die bij de functieaanroepen horen.

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

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

Vergelijkbaar met hoe we de functienamen hebben behandeld, bergen we de argumenten op in een geïndexeerde array.

Vervolgens kijken we naar normale gebruikersberichten, die één token tegelijk van de server zullen binnenkomen en worden toegewezen aan de `new_content` variabele. We moeten ook `finish_reason` in de gaten houden. Deze zal `nil` blijven tot het laatste deel van de uitvoerreeks.

```
  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
```

Belangrijk is dat we een patroonherkenningsexpressie toevoegen om foutmeldingen van

de AI-modelprovider af te handelen. In lokale ontwikkelomgevingen gooien we een exceptie, maar in productie loggen we de fout en ronden we af.

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

De laatste else-clausule van case wordt uitgevoerd als geen van de voorgaande patronen overeenkwamen. Het is slechts een veiligheidsmaatregel, zodat we het ontdekken als het AI-model ons onherkenbare fragmenten begint te sturen.

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

De `send_to_client` methode is verantwoordelijk voor het verzenden van de gebufferde inhoud naar de client. Deze controleert of de buffer niet leeg is, werkt de inhoud van het botbericht bij, geeft het botbericht weer en slaat de inhoud op in de database om gegevenspersistentie te waarborgen.

```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
```

De `finalize`-methode wordt aangeroepen wanneer de stream processing is voltooid. Deze verwerkt de functie-aanroepen als die tijdens de stream zijn ontvangen, werkt het botbericht bij met de definitieve inhoud en andere relevante informatie, en reset de functie-aanroepgeschiedenis

```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
```

Als het model besluit om een functie aan te roepen, moet je die functieaanroep (naam

en argumenten) zodanig "afhandelen" dat deze wordt uitgevoerd en er `function_call` en `function_result` berichten worden toegevoegd aan het gespreksverslag

Uit mijn ervaring is het beter om het aanmaken van functieberichten op één plek in je codebase af te handelen, in plaats van te vertrouwen op de implementaties van de tools. Dit is niet alleen netter, maar heeft ook een zeer belangrijke praktische reden: als het AI-model een functie aanroept en de resulterende aanroep- en resultaatberichten niet in het transcript ziet tijdens het doorlopen, *zal het dezelfde functie opnieuw aanroepen*. Mogelijk tot in het oneindige. Onthoud dat de AI volledig toestandsloos is, dus tenzij je die functieaanroepen terugkoppelt, zijn ze voor het model nooit gebeurd.

```
# 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
```

Het wissen van de functie-aanroepgeschiedenis na het verzenden is net zo belangrijk als ervoor zorgen dat de aanroep en resultaten in je transcript terechtkomen, zodat je niet steeds dezelfde functies blijft aanroepen elke keer dat je de lus doorloopt.

De "Conversatielus"

I Ik blijf het hebben over lussen, maar als je nieuw bent met functie-aanroepen, is het misschien niet direct duidelijk *waarom* we een lus nodig hebben. De reden is dat zodra de AI je vraagt om toolfuncties namens haar uit te voeren, ze stopt met antwoorden.

Het is aan jou om die functies uit te voeren, de resultaten te verzamelen, de resultaten aan het transcript toe te voegen, en vervolgens de oorspronkelijke prompt opnieuw in te dienen om een nieuwe set functie-aanroepen of gebruikersgerichte resultaten te krijgen.

In de `PromptSubscriber` klasse gebruiken we de `prompt` methode van de `PromptDeclarations` module om het gedrag van de conversatielus te definiëren. De `until` parameter is ingesteld op `-> { bot_message.complete? }`, wat betekent dat de lus doorgaat totdat het `bot_message` als voltooid is gemarkeerd.

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

Maar wanneer wordt `bot_message` als voltooid gemarkeerd? Als je het bent vergeten, kijk dan terug naar regel 13 van de `finalize` methode.

Laten we de volledige streamverwerkingslogica doornemen.

1. De `PromptSubscriber` ontvangt een nieuw gebruikersbericht via de `message_created` methode, die wordt aangeroepen door het Wisper publicatie/abonnement-systeem telkens wanneer de eindgebruiker een nieuwe prompt maakt.
2. De `prompt` klassemethode definieert op declaratieve wijze het gedrag van de chatafronding-logica voor de `PromptSubscriber`. Het AI-model zal een chatafronding uitvoeren met de berichtinhoud van de gebruiker, een nieuwe instantie van `ReplyStream` als de streamparameter, en de gespecificeerde lusvoorwaarde.
3. Het AI-model verwerkt de prompt en begint met het genereren van een antwoord. Terwijl het antwoord wordt gestreamd, wordt de `call` methode van de `ReplyStream` instantie aangeroepen voor elk deel van de data.

4. Als het AI-model besluit om een hulpfunctie aan te roepen, worden de functienaam en argumenten uit het deel geëxtraheerd en respectievelijk opgeslagen in de `f_name` en `f_arguments` arrays.
5. Als het AI-model gebruikersgerichte inhoud genereert, wordt deze gebufferd en naar de client verzonden via de `send_to_client` methode.
6. Zodra de streamverwerking voltooid is, wordt de `finalize` methode aangeroepen. Als er tijdens de stream hulpfuncties zijn aangeroepen, worden deze afgehandeld met behulp van de `dispatch` methode van de `PromptSubscriber`.
7. De `dispatch` methode voegt een `function_call` bericht toe aan het gespreksverslag, voert de corresponderende hulpfunctie uit, en voegt een `function_result` bericht toe aan het verslag met het resultaat van de functieaanroep.
8. Na het afhandelen van de hulpfuncties wordt de functieaanroepgeschiedenis gewist om dubbele functieaanroepen in volgende lussen te voorkomen.
9. Als er geen hulpfuncties zijn aangeroepen, werkt de `finalize` methode het `bot_message` bij met de definitieve inhoud, markeert het als voltooid, en stuurt het bijgewerkte bericht naar de client.
10. De lusvoorwaarde `-> { bot_message.complete? }` wordt geëvalueerd. Als het `bot_message` niet als voltooid is gemarkeerd, gaat de lus door en wordt de originele prompt opnieuw ingediend met het bijgewerkte gespreksverslag.
11. Stappen 3-10 worden herhaald totdat het `bot_message` als voltooid is gemarkeerd, wat aangeeft dat het AI-model klaar is met het genereren van zijn antwoord en er geen verdere hulpfuncties hoeven te worden uitgevoerd.

Door deze conversatielus te implementeren, stel je het AI-model in staat om een wisselwerking met de applicatie aan te gaan, hulpfuncties uit te voeren wanneer nodig en gebruikersgerichte antwoorden te genereren totdat het gesprek een natuurlijke conclusie bereikt.

De combinatie van streamverwerking en de conversatielus maakt dynamische en interactieve AI-gestuurde ervaringen mogelijk, waarbij het AI-model gebruikersinvoer

kan verwerken, verschillende hulpmiddelen en functies kan gebruiken, en realtime antwoorden kan geven op basis van de zich ontwikkelende gesprekscontext.

Automatische Voortzetting

Het is belangrijk om je bewust te zijn van AI-uitvoerbeperkingen. De meeste modellen hebben een maximaal aantal tokens dat ze in één antwoord kunnen genereren, wat wordt bepaald door de `max_tokens` parameter. Als het AI-model deze limiet bereikt tijdens het genereren van een antwoord, zal het abrupt stoppen en aangeven dat de uitvoer is afgekapt.

In de streaming-respons van de AI-platform API kun je deze situatie detecteren door de `finish_reason` variabele in het deel te onderzoeken. Als de `finish_reason` is ingesteld op `"length"` (of een andere sleutelwaarde specifiek voor het model), betekent dit dat het model zijn maximale tokenlimiet heeft bereikt tijdens het genereren en de uitvoer voortijdig is afgebroken.

Een manier om dit scenario elegant af te handelen en een naadloze gebruikerservaring te bieden, is door een automatisch voortzettingsmechanisme te implementeren in je streamverwerkingslogica. Door een patroonherkenning toe te voegen voor lengtegebonden afsluitredenen, kun je ervoor kiezen om te lussen en de uitvoer automatisch voort te zetten vanaf waar deze was gebleven.

Hier is een opzettelijk vereenvoudigd voorbeeld van hoe je de `call` methode in de `ReplyStream` klasse kunt aanpassen om automatische voortzetting te ondersteunen:

```
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
```

In deze aangepaste versie, wanneer de `finish_reason` afgekapte output aangeeft, voegen we, in plaats van de stream te finaliseren, een paar berichten toe aan het transcript zonder te finaliseren, verplaatsen we het originele gebruikersgerichte antwoordbericht naar de “onderkant” van het transcript door zijn `created_at` attribuut bij te werken, en laten we vervolgens de lus gebeuren, zodat de AI doorgaat met genereren waar deze was gestopt.

Onthoud dat het AI-voltooiingseindpunt statusloos is. Het "weet" alleen wat je het vertelt via het transcript. In dit geval is de manier waarop we aan de AI communiceren dat deze werd afgekapt door het toevoegen van "onzichtbare" (voor de eindgebruiker) berichten aan het transcript. Onthoud echter dat dit een opzettelijk vereenvoudigd voorbeeld is. Een echte implementatie zou verder transcriptbeheer moeten uitvoeren om ervoor te zorgen dat we geen tokens verspillen en/of de AI niet in verwarring brengen met gedupliceerde assistent-berichten in het transcript.

Een echte implementatie van automatische voortzetting zou ook zogenaamde "circuitonderbreker-logica" moeten bevatten om ongecontroleerde lussen te voorkomen. De reden hiervoor is dat, gegeven bepaalde soorten gebruikersprompts en lage `max_tokens` instellingen, de AI eindeloos zou kunnen doorgaan met het genereren van gebruikersgerichte output.

> Houd er rekening mee dat elke lus een apart verzoek vereist, en dat elk verzoek je hele transcript opnieuw verbruikt. Je moet zeker de afweging maken tussen gebruikerservaring en API-gebruik bij het beslissen of je automatische voortzetting in je applicatie wilt implementeren. Automatische voortzetting kan in het bijzonder gevaarlijk duur zijn, vooral bij gebruik van premium commerciële modellen.

Conclusie

Streamverwerking is een kritiek aspect van het bouwen van AI-aangedreven applicaties die toolgebruik combineren met live AI-responses. Door het efficiënt afhandelen van de streaming data van AI-platform API's, kun je een naadloze en interactieve gebruikerservaring bieden, grote responses afhandelen, bronnengebruik optimaliseren en fouten elegant afhandelen.

De aangeboden `Conversation::ReplyStream` klasse demonstreert hoe streamverwerking kan worden geïmplementeerd in een Ruby-applicatie met behulp van patroonherkenning en gebeurtenisgestuurde architectuur. Door streamverwerkingstechnieken te begrijpen en te benutten, kun je het volledige potentieel van AI-integratie in je applicaties ontsluiten en krachtige en boeiende gebruikerservaringen leveren.

Zelfherstellende Data

Zelfherstellende data is een krachtige benadering om data-integriteit, consistentie en kwaliteit in applicaties te waarborgen door gebruik te maken van de mogelijkheden van grote taalmodellen (LLM's). Deze categorie patronen richt zich op het idee om AI te gebruiken voor het automatisch detecteren, diagnosticeren en corrigeren van data-anomalieën, inconsistenties of fouten, waardoor de last voor ontwikkelaars wordt verminderd en een hoog niveau van databelrouwbaarheid wordt gehandhaafd.

In de kern erkennen de zelfherstellende datapatronen dat data de levensader is van elke applicatie, en het waarborgen van de nauwkeurigheid en integriteit ervan is cruciaal voor het goed functioneren en de gebruikerservaring van de applicatie. Het beheren en onderhouden van datakwaliteit kan echter een complexe en tijdrovende taak zijn, vooral naarmate applicaties groeien in omvang en complexiteit. Hier komt de kracht van AI in beeld.

In de zelfherstellende datapatronen worden AI-workers ingezet om continu de data van uw applicatie te monitoren en te analyseren. Deze modellen hebben het vermogen om patronen, relaties en anomalieën binnen de data te begrijpen en te interpreteren. Door gebruik te maken van hun natuurlijke taalverwerking en begrip kunnen ze potentiële problemen of inconsistenties in de data identificeren en passende acties ondernemen om deze te herstellen.

Het proces van zelfherstellende data omvat meestal verschillende belangrijke stappen:

1. **Datamonitoring**: AI-workers monitoren constant de datastromen, databases of opslagsystemen van de applicatie, op zoek naar tekenen van anomalieën, inconsistenties of fouten. Als alternatief kunt u een AI-component activeren als reactie op een uitzondering.
2. **Anomaliedetectie**: Wanneer een probleem wordt gedetecteerd, analyseert de AI-worker de data in detail om de specifieke aard en omvang van het probleem te identificeren. Dit kan gaan om het detecteren van ontbrekende waarden, inconsistente formaten of data die vooraf gedefinieerde regels of beperkingen schendt.
3. **Diagnose en Correctie**: Zodra het probleem is geïdentificeerd, gebruikt de AI-worker zijn kennis en begrip van het datadomein om de juiste aanpak te bepalen. Dit kan betekenen dat de data automatisch wordt gecorrigeerd, ontbrekende waarden worden ingevuld, of het probleem wordt gemarkeerd voor menselijke interventie indien nodig.
4. **Continu Leren (optioneel**, **afhankelijk van gebruik)**: Terwijl uw AI-worker verschillende dataproblemen tegenkomt en oplost, kan deze output genereren die beschrijft wat er is gebeurd en hoe erop is gereageerd. Deze metadata kan worden gebruikt in leerprocessen waardoor u (en mogelijk het onderliggende model, via fine-tuning) effectiever en efficiënter wordt in het identificeren en oplossen van data-anomalieën.

Door automatisch dataproblemen te detecteren en te corrigeren, kunt u ervoor

zorgen dat uw applicatie werkt met kwalitatief hoogwaardige, betrouwbare data. Dit vermindert het risico dat fouten, inconsistenties of data-gerelateerde bugs de functionaliteit of gebruikerservaring van de applicatie beïnvloeden.

Zodra u AI-workers heeft die de taak van datamonitoring en -correctie uitvoeren, kunt u zich richten op andere kritieke aspecten van de applicatie. Dit bespaart tijd en middelen die anders zouden worden besteed aan handmatige dataopschoning en onderhoud. Sterker nog, naarmate uw applicaties groeien in omvang en complexiteit, wordt het handmatig beheren van datakwaliteit steeds uitdagender. De "Zelfherstellende Data" patronen schalen effectief door de kracht van AI te benutten om grote hoeveelheden data te verwerken en problemen in realtime te detecteren.

Vanwege hun aard kunnen AI-modellen zich met weinig tot geen toezicht aanpassen aan veranderende datapatronen, schema's of vereisten. Zolang hun richtlijnen adequate begeleiding bieden, vooral met betrekking tot beoogde resultaten, kan uw applicatie mogelijk evolueren en nieuwe datascenario's afhandelen zonder uitgebreide handmatige interventie of codewijzigingen.

De zelfherstellende datapatronen sluiten goed aan bij de andere categorieën patronen die we hebben besproken, zoals "Veelheid aan Workers". De zelfherstellende datacapaciteit kan worden gezien als een gespecialiseerd soort worker die zich specifiek richt op het waarborgen van datakwaliteit en -integriteit. Dit soort worker werkt samen met andere AI-workers, waarbij elk bijdraagt aan verschillende aspecten van de functionaliteit van de applicatie.

Het implementeren van zelfherstellende datapatronen in de praktijk vereist zorgvuldig ontwerp en integratie van AI-modellen in de applicatiearchitectuur. Vanwege de risico's van dataverlies en -corruptie moet u duidelijke richtlijnen definiëren voor hoe u deze techniek zult gebruiken. U moet ook rekening houden met factoren zoals prestaties, schaalbaarheid en databeveiliging.

Praktijkvoorbeeld: Het Repareren van Beschadigde JSON

Een van de meest praktische en handige manieren om zelfherstellende data te benutten is ook heel eenvoudig uit te leggen: het repareren van beschadigde JSON.

Deze techniek kan worden toegepast op de veel voorkomende uitdaging van het omgaan met imperfecte of inconsistente data gegenereerd door LLM's, zoals beschadigde JSON, en biedt een aanpak voor het automatisch detecteren en corrigeren van deze problemen.

Bij Olympia kom ik regelmatig scenario's tegen waarbij LLM's JSON-data genereren die niet volledig geldig is. Dit kan verschillende oorzaken hebben, zoals wanneer het LLM commentaar toevoegt voor of na de eigenlijke JSON-code, of syntaxisfouten introduceert zoals ontbrekende komma's of niet-geëscapete dubbele aanhalingstekens. Deze problemen kunnen leiden tot parse-fouten en verstoringen veroorzaken in de functionaliteit van de applicatie.

Om dit probleem aan te pakken, heb ik een praktische oplossing geïmplementeerd in de vorm van een JsonFixer-klasse. Deze klasse belichaamt het "Self-Healing Data" patroon door de beschadigde JSON als invoer te nemen en een LLM te gebruiken om deze te repareren, waarbij zoveel mogelijk informatie en intentie behouden blijft.

```
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
```

Merk op hoe `JsonFixer` Ventriloquist gebruikt om de AI-responses te sturen.

Het proces van zelfherstellende JSON-data werkt als volgt:

1. **JSON-generatie**: Een LLM wordt gebruikt om JSON-data te genereren op basis van bepaalde prompts of vereisten. Echter, door de aard van LLMs is de gegenereerde JSON niet altijd volledig geldig. De JSON-parser zal uiteraard een `ParserError` genereren als je ongeldige JSON aanlevert.

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

Merk op dat het foutbericht ook wordt doorgegeven aan de `JSONFixer`-aanroep, zodat deze niet volledig hoeft aan te nemen wat er mis is met de data, vooral omdat de parser vaak precies aangeeft wat er mis is.

2. **LLM-gebaseerde Correctie**: De `JSONFixer`-klasse stuurt de beschadigde JSON terug naar een LLM, samen met een specifieke prompt of instructie om de JSON te

repareren waarbij de originele informatie en bedoeling zoveel mogelijk behouden blijven. De LLM, getraind op enorme hoeveelheden data en met begrip van JSON-syntax, probeert de fouten te corrigeren en een geldige JSON-string te genereren. Responsbegrenzing wordt gebruikt om de output van de LLM te beperken, en we kiezen Mixtral 8x7B als het AI-model, aangezien het bijzonder geschikt is voor dit soort taken.

3. **Validatie en Integratie**: De gerepareerde JSON-string die door de LLM wordt teruggegeven, wordt verwerkt door de `JSONFixer`-klasse zelf, omdat deze `chat_completion(json: true)` aanroept. Als de gerepareerde JSON de validatie doorstaat, wordt deze geïntegreerd in de werkstroom van de applicatie, waardoor de applicatie naadloos door kan gaan met het verwerken van de data. De slechte JSON is "genezen".

Hoewel ik mijn eigen `JSONFixer`-implementatie meerdere keren heb geschreven en herschreven, betwijfel ik of de totale tijd die in al die versies is geïnvesteerd meer dan een uur of twee bedraagt.

Merk op dat het behoud van de oorspronkelijke bedoeling een kernelement is van elk zelfherstellend datapatroon. Het LLM-gebaseerde correctieproces streeft ernaar om de originele informatie en bedoeling van de gegenereerde JSON zoveel mogelijk te behouden. Dit zorgt ervoor dat de gerepareerde JSON zijn semantische betekenis behoudt en effectief kan worden gebruikt binnen de context van de applicatie.

Deze praktische implementatie van de "Zelfherstellende Data"-aanpak in Olympia laat duidelijk zien hoe AI, en specifiek LLM's, kunnen worden ingezet om praktische data-uitdagingen op te lossen. Het toont de kracht van het combineren van traditionele programmeertechnieken met AI-mogelijkheden om robuuste en efficiënte applicaties te bouwen.

Postel's Law en het "Zelfherstellende Data"-Patroon

"Zelfherstellende Data", zoals geïllustreerd door de JSONFixer-klasse, sluit goed aan bij het principe bekend als Postel's Law, ook wel het Robuustheidsprincipe genoemd. Postel's Law stelt:

"Wees conservatief in wat je doet, wees liberaal in wat je accepteert van anderen."

Dit principe, oorspronkelijk geformuleerd door Jon Postel, een pionier van het vroege internet, benadrukt het belang van het bouwen van systemen die tolerant zijn voor diverse of zelfs licht incorrecte invoer, terwijl ze strikt vasthouden aan gespecificeerde protocollen bij het verzenden van uitvoer.

In de context van "Zelfherstellende Data" belichaamt de JSONFixer-klasse Postel's Law door liberaal te zijn in het accepteren van beschadigde of imperfecte JSON-data gegenereerd door LLM's. Het verwerpt of faalt niet onmiddellijk wanneer het JSON tegenkomt die niet strikt voldoet aan het verwachte formaat. In plaats daarvan neemt het een tolerante aanpak en probeert het de JSON te repareren met behulp van de kracht van LLM's.

Door liberaal te zijn in het accepteren van imperfecte JSON, demonstreert de JSONFixer-klasse robuustheid en flexibiliteit. Het erkent dat data in de echte wereld vaak in verschillende vormen komt en niet altijd aan strikte specificaties voldoet. Door deze afwijkingen elegant af te handelen en te corrigeren, zorgt de klasse ervoor dat de applicatie soepel kan blijven functioneren, zelfs in aanwezigheid van imperfecte data.

Aan de andere kant houdt de JSONFixer-klasse zich ook aan het conservatieve aspect van Postel's Law als het gaat om de uitvoer. Na het repareren van de JSON met behulp van LLM's, valideert de klasse de gecorrigeerde JSON om ervoor te zorgen dat deze strikt voldoet aan het verwachte formaat. Het behoudt de integriteit en correctheid

van de data voordat deze wordt doorgegeven aan andere delen van de applicatie. Deze conservatieve aanpak garandeert dat de uitvoer van de JSONFixer-klasse betrouwbaar en consistent is, wat interoperabiliteit bevordert en de verspreiding van fouten voorkomt.

Interessante weetjes over Jon Postel:

- Jon Postel (1943-1998) was een Amerikaanse informaticus die een cruciale rol speelde in de ontwikkeling van het internet. Hij stond bekend als de "God van het Internet" vanwege zijn belangrijke bijdragen aan de onderliggende protocollen en standaarden.
- Postel was de redacteur van de Request for Comments (RFC)-documentreeks, een serie technische en organisatorische notities over het internet. Hij schreef of co-schreef meer dan 200 RFC's, waaronder de fundamentele protocollen zoals TCP, IP en SMTP.
- Naast zijn technische bijdragen stond Postel bekend om zijn bescheiden en samenwerkende aanpak. Hij geloofde in het belang van het bereiken van consensus en samenwerken om een robuust en interoperabel netwerk te bouwen.
- Postel diende als Directeur van de Computer Networks Division bij het Information Sciences Institute (ISI) van de University of Southern California (USC) van 1977 tot aan zijn vroegtijdige dood in 1998.
- Als erkenning voor zijn enorme bijdragen werd Postel postuum onderscheiden met de prestigieuze Turing Award in 1998, vaak aangeduid als de "Nobelprijs voor Informatica."

De `JSONFixer`-klasse bevordert robuustheid, flexibiliteit en interoperabiliteit, wat kernwaarden waren die Postel gedurende zijn hele carrière voorstond. Door systemen te bouwen die tolerant zijn voor onvolkomenheden, terwijl ze zich strikt aan protocollen houden, kunnen we toepassingen creëren die veerkrachtiger en aanpasbaarder zijn bij praktische uitdagingen.

Overwegingen en Contra-indicaties

De toepasbaarheid van zelfherstellende data-aanpakken is volledig afhankelijk van het soort gegevens dat je applicatie verwerkt. Er is een reden waarom je mogelijk niet zomaar `JSON.parse` wilt aanpassen om automatisch *alle JSON-parsing fouten* in je applicatie te corrigeren: niet alle fouten kunnen of moeten automatisch worden gecorrigeerd.

Zelfherstel is bijzonder complex wanneer het gekoppeld wordt aan regelgevings- of nalevingsvereisten met betrekking tot gegevensverwerking en -verwerking. Sommige sectoren, zoals de gezondheidszorg en financiële sector, hebben zodanig strenge voorschriften met betrekking tot data-integriteit en controleerbaarheid dat het uitvoeren van "black box" datacorrecties zonder adequaat toezicht of logging in strijd kan zijn met deze voorschriften. Het is cruciaal om ervoor te zorgen dat alle zelfherstellende datatechnieken die je ontwikkelt, in overeenstemming zijn met de toepasselijke wettelijke en regelgevende kaders.

Het toepassen van zelfherstellende datatechnieken, vooral die met AI-modellen, kan ook grote invloed hebben op de prestaties en het brongebruik van applicaties. Het verwerken van grote hoeveelheden gegevens via AI-modellen voor foutdetectie en -correctie kan rekenintensief zijn. Het is belangrijk om de afweging te maken tussen de voordelen van zelfherstellende data en de bijbehorende prestatie- en resourcekosten.

Dat gezegd hebbende, laten we eens kijken naar de factoren die een rol spelen bij het beslissen wanneer en waar deze krachtige aanpak toe te passen.

Datakritikaliteit

Bij het overwegen van de toepassing van zelfherstellende datatechnieken is het cruciaal om de kritikaliteit van de te verwerken gegevens te beoordelen. Het kritikaliteitsniveau verwijst naar het belang en de gevoeligheid van de gegevens in de context van je applicatie en het bedrijfsdomein.

In sommige gevallen is het automatisch corrigeren van datafouten mogelijk niet gepast, vooral als de gegevens zeer gevoelig zijn of juridische implicaties hebben. Overweeg bijvoorbeeld de volgende scenario's:

1. **Financiële Transacties**: In financiële applicaties, zoals banksystemen of handelsplatformen, is data-nauwkeurigheid van het grootste belang. Zelfs kleine fouten in financiële gegevens kunnen belangrijke gevolgen hebben, zoals onjuiste rekeningsaldi, verkeerd geleide gelden of foutieve handelsbeslissingen. In deze gevallen kunnen geautomatiseerde correcties zonder grondige verificatie en controle onaanvaardbare risico's met zich meebrengen.
2. **Medische Dossiers**: Zorgtoepassingen werken met zeer gevoelige en vertrouwelijke patiëntgegevens. Onnauwkeurigheden in medische dossiers kunnen ernstige gevolgen hebben voor de veiligheid van patiënten en behandelingsbeslissingen. Het automatisch wijzigen van medische gegevens zonder adequaat toezicht en validatie door gekwalificeerde zorgprofessionals kan in strijd zijn met regelgevingsvereisten en de gezondheid van patiënten in gevaar brengen.
3. **Juridische Documenten**: Toepassingen die juridische documenten verwerken, zoals contracten, overeenkomsten of rechtbankdocumenten, vereisen strikte nauwkeurigheid en integriteit. Zelfs kleine fouten in juridische gegevens kunnen belangrijke juridische gevolgen hebben. Geautomatiseerde correcties zijn in dit domein mogelijk niet gepast, aangezien de gegevens vaak handmatige controle en verificatie door juridische experts vereisen om de geldigheid en afdwingbaarheid te waarborgen.

In deze kritieke datascenario's wegen de risico's van geautomatiseerde correcties vaak niet op tegen de potentiële voordelen. De gevolgen van het introduceren van fouten of het incorrect wijzigen van gegevens kunnen ernstig zijn, wat kan leiden tot financiële verliezen, juridische aansprakelijkheid of zelfs schade aan personen.

Bij het werken met zeer kritieke gegevens is het essentieel om prioriteit te geven aan handmatige verificatie- en validatieprocessen. Menselijk toezicht en expertise zijn cruciaal voor het waarborgen van de nauwkeurigheid en integriteit van de gegevens. Geautomatiseerde zelfherstellende technieken kunnen nog steeds worden gebruikt om mogelijke fouten of inconsistenties te markeren, maar de uiteindelijke beslissing over correcties moet menselijk oordeel en goedkeuring omvatten.

Het is echter belangrijk op te merken dat niet alle gegevens in een applicatie hetzelfde kritikaliteitsniveau hebben. Binnen dezelfde applicatie kunnen er subsets van gegevens zijn die minder gevoelig zijn of waarbij fouten minder impact hebben. In dergelijke gevallen kunnen zelfherstellende datatechnieken selectief worden toegepast op die specifieke datasubsets, terwijl kritieke gegevens onderworpen blijven aan handmatige verificatie.

De sleutel is om zorgvuldig de kritikaliteit van elke datacategorie in je applicatie te beoordelen en duidelijke richtlijnen en processen te definiëren voor het afhandelen van correcties op basis van de bijbehorende risico's en implicaties. Door onderscheid te maken tussen kritieke (zoals grootboeken, medische dossiers) en niet-kritieke gegevens (zoals mailingadressen, bronwaarschuwingen), kun je een balans vinden tussen het benutten van de voordelen van zelfherstellende datatechnieken waar gepast en het handhaven van strikte controle en toezicht waar nodig.

Uiteindelijk moet de beslissing om zelfherstellende datatechnieken toe te passen op kritieke gegevens worden genomen in overleg met domeinexperts, juridisch adviseurs en andere relevante belanghebbenden. Het is essentieel om rekening te houden met de specifieke vereisten, voorschriften en risico's die verbonden zijn aan de gegevens van je applicatie en de datacorrectiestrategieën daarop af te stemmen.

Ernst van Fouten

Bij het toepassen van zelfherstellende datatechnieken is het belangrijk om de ernst en impact van de datafouten te beoordelen. Niet alle fouten zijn gelijk, en de juiste aanpak

kan variëren afhankelijk van de ernst van het probleem.

Kleine inconsistenties of opmaakproblemen kunnen geschikt zijn voor automatische correctie. Bijvoorbeeld, een zelfherstellende dataverwerker die bedoeld is om defecte JSON te repareren kan ontbrekende komma's of niet-geëscapete dubbele aanhalingstekens afhandelen zonder de betekenis of structuur van de gegevens significant te wijzigen. Dit soort fouten zijn vaak eenvoudig te corrigeren en hebben minimale impact op de algehele data-integriteit.

Echter, ernstigere fouten die de betekenis of integriteit van de data fundamenteel veranderen, vereisen mogelijk een andere aanpak. In dergelijke gevallen zijn geautomatiseerde correcties mogelijk niet toereikend, en kan menselijke tussenkomst noodzakelijk zijn om de nauwkeurigheid en validiteit van de data te waarborgen.

Dit is waar het concept om AI zelf te gebruiken voor het bepalen van de ernst van fouten in beeld komt. Door gebruik te maken van de mogelijkheden van AI-modellen, kunnen we zelfherstellende data-workers ontwerpen die niet alleen fouten corrigeren, maar ook de ernst van deze fouten beoordelen en weloverwogen beslissingen nemen over hoe ermee om te gaan.

Laten we bijvoorbeeld een zelfherstellende data-worker beschouwen die verantwoordelijk is voor het corrigeren van inconsistenties in data die een klantendatabase binnenstroomt. De worker kan worden ontworpen om de data te analyseren en potentiële fouten te identificeren, zoals ontbrekende of tegenstrijdige informatie. In plaats van alle fouten automatisch te corrigeren, kan de worker worden uitgerust met aanvullende gereedschapsaanroepen die het mogelijk maken om ernstige fouten te markeren voor menselijke beoordeling.

Hier is een voorbeeld van hoe dit geïmplementeerd kan worden:

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

  attr_accessor :customer

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

  def initialize(customer)
    self.customer = customer
  end

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

        < additional instructions here... >

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

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

    self.stop = ["```"]

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

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

In dit voorbeeld is de `CustomerDataHealer` worker ontworpen om inconsistenties in klantgegevens te identificeren en te corrigeren. Opnieuw gebruiken we Response Fencing en Ventriloquist om gestructureerde output te krijgen. Belangrijk is dat de

systeemrichtlijn van de worker instructies bevat om de `flag_for_review` functie te gebruiken als er ernstige fouten worden aangetroffen.

Wanneer de worker de klantgegevens verwerkt, analyseert deze de data en probeert eventuele inconsistenties te corrigeren. Als de worker vaststelt dat de fouten ernstig zijn en menselijke interventie vereisen, kan deze de `flag_for_review` tool gebruiken om de gegevens te markeren en een reden voor de markering te verstrekken.

De `chat_completion` methode wordt aangeroepen met `json: true` om de gecorrigeerde klantgegevens als JSON te verwerken. Er is geen voorziening voor het maken van een lus na een functie-aanroep, dus het resultaat zal leeg zijn als `flag_for_review` werd aangeroepen. Anders wordt de klant bijgewerkt met de beoordeelde en mogelijk gecorrigeerde gegevens.

Door beoordeling van de ernst van fouten en de mogelijkheid om gegevens te markeren voor menselijke controle op te nemen, wordt de zelfherstellende data worker intelligenter en aanpasbaarder. Deze kan kleine fouten automatisch afhandelen terwijl ernstige fouten worden geëscaleerd naar menselijke experts voor handmatige interventie.

De specifieke criteria voor het bepalen van de ernst van fouten kunnen worden gedefinieerd in de richtlijn van de worker op basis van domeinkennis en bedrijfsvereisten. Factoren zoals de impact op data-integriteit, het risico op gegevensverlies of -corruptie, en de gevolgen van onjuiste gegevens kunnen worden meegewogen bij het beoordelen van de ernst.

Door AI te gebruiken voor het beoordelen van de ernst van fouten en opties te bieden voor menselijke interventie, kunnen zelfherstellende datatechnieken een balans vinden tussen automatisering en het behoud van gegevensnauwkeurigheid. Deze aanpak zorgt ervoor dat kleine fouten efficiënt worden gecorrigeerd terwijl ernstige fouten de nodige aandacht en expertise krijgen van menselijke beoordelaars.

Domein Complexiteit

Bij het overwegen van de toepassing van zelfherstellende datatechnieken is het belangrijk om de complexiteit van het datadomein en de regels die de structuur en relaties ervan bepalen te evalueren. De complexiteit van het domein kan een aanzienlijke invloed hebben op de effectiviteit en haalbaarheid van geautomatiseerde datacorrectie-benaderingen.

Zelfherstellende datatechnieken werken goed wanneer de gegevens duidelijk gedefinieerde patronen en beperkingen volgen. In domeinen waar de datastructuur relatief eenvoudig is en de relaties tussen data-elementen overzichtelijk zijn, kunnen geautomatiseerde correcties met een hoge mate van vertrouwen worden toegepast. Het corrigeren van formatteringsproblemen of het afdwingen van basis datatype-beperkingen kan bijvoorbeeld vaak effectief worden afgehandeld door zelfherstellende data workers.

Echter, naarmate de complexiteit van het datadomein toeneemt, groeien ook de uitdagingen die gepaard gaan met geautomatiseerde datacorrectie. In domeinen met ingewikkelde bedrijfslogica, complexe relaties tussen data-entiteiten, of domeinspecifieke regels en uitzonderingen, kunnen zelfherstellende datatechnieken niet altijd de nuances vastleggen en kunnen ze onbedoelde gevolgen introduceren.

Laten we een voorbeeld nemen van een complex domein: een financieel handelssysteem. In dit domein omvatten de gegevens verschillende financiële instrumenten, marktgegevens, handelsregels en regelgevingsvereisten. De relaties tussen verschillende data-elementen kunnen ingewikkeld zijn, en de regels die de geldigheid en consistentie van gegevens bepalen kunnen zeer specifiek zijn voor het domein.

In een dergelijk complex domein zou een zelfherstellende data worker die belast is met het corrigeren van inconsistenties in handelsgegevens een diepgaand begrip moeten hebben van de domeinspecifieke regels en beperkingen. Deze zou rekening moeten houden met factoren zoals marktregulering, handelslimieten, risicoberekeningen en

afwikkelingsprocedures. Geautomatiseerde correcties kunnen in deze context niet altijd de volledige complexiteit van het domein vastleggen en kunnen onbedoeld fouten introduceren of domeinspecifieke regels schenden.

Om de uitdagingen van domeincomplexiteit aan te pakken, kunnen zelfherstellende datatechnieken worden verbeterd door domeinspecifieke kennis en regels op te nemen in de AI-modellen en workers. Dit kan worden bereikt door technieken zoals:

1. **Domeinspecifieke Training**: De AI-modellen die worden gebruikt voor zelfherstellende data kunnen worden gestuurd of zelfs verfijnd op domeinspecifieke datasets die de complexiteit en regels van het specifieke domein vastleggen. Door de modellen bloot te stellen aan representatieve gegevens en scenario's, kunnen ze de patronen, beperkingen en uitzonderingen leren die specifiek zijn voor het domein.
2. **Regelgebaseerde Beperkingen**: Zelfherstellende data workers kunnen worden uitgebreid met expliciete regelgebaseerde beperkingen die domeinspecifieke kennis coderen. Deze regels kunnen worden gedefinieerd door domeinexperts en geïntegreerd in het datacorrectieproces. De AI-modellen kunnen deze regels dan gebruiken om hun beslissingen te sturen en naleving van domeinspecifieke vereisten te waarborgen.
3. **Samenwerking met Domeinexperts**: In complexe domeinen is het cruciaal om domeinexperts te betrekken bij het ontwerp en de ontwikkeling van zelfherstellende datatechnieken. Domeinexperts kunnen waardevolle inzichten verschaffen in de complexiteit van de gegevens, de bedrijfsregels en de mogelijke randgevallen. Hun kennis kan worden opgenomen in de AI-modellen en workers om de nauwkeurigheid en betrouwbaarheid van geautomatiseerde datacorrecties te verbeteren met behulp van Human In The Loop patronen.
4. **Incrementele en Iteratieve Aanpak**: Bij het omgaan met complexe domeinen is het vaak gunstig om een incrementele en iteratieve aanpak voor zelfherstellende data te hanteren. In plaats van te proberen correcties voor het hele domein in

één keer te automatiseren, focus je op specifieke subdomeinen of datacategorieën waar de regels en beperkingen goed begrepen worden. Breid de reikwijdte van zelfherstellende technieken geleidelijk uit naarmate het begrip van het domein groeit en de technieken effectief blijken te zijn.

Door rekening te houden met de complexiteit van het datadomein en domeinspecifieke kennis te integreren in zelfherstellende datatechnieken, kun je een balans vinden tussen automatisering en nauwkeurigheid. Het is belangrijk om te erkennen dat zelfherstellende data geen universele oplossing is en dat de aanpak moet worden afgestemd op de specifieke vereisten en uitdagingen van elk domein.

In complexe domeinen kan een hybride aanpak die zelfherstellende datatechnieken combineert met menselijke expertise en toezicht het meest effectief zijn. Geautomatiseerde correcties kunnen routinematige en goed gedefinieerde gevallen afhandelen, terwijl complexe scenario's of uitzonderingen kunnen worden gemarkeerd voor menselijke beoordeling en interventie. Deze samenwerkende aanpak zorgt ervoor dat de voordelen van automatisering worden gerealiseerd terwijl de noodzakelijke controle en nauwkeurigheid in complexe datadomeinen behouden blijven.

Verklaarbaarheid en Transparantie

Verklaarbaarheid verwijst naar het vermogen om de redenering achter de beslissingen van AI-modellen te begrijpen en te interpreteren, terwijl transparantie gaat over het bieden van duidelijk inzicht in het datacorrectieproces.

In veel contexten moeten datawijzigingen controleerbaar en verantwoordbaar zijn. Belanghebbenden, waaronder zakelijke gebruikers, auditors en regelgevende instanties, kunnen uitleg nodig hebben over waarom bepaalde datacorrecties zijn uitgevoerd en hoe de AI-modellen tot die beslissingen zijn gekomen. Dit is vooral cruciaal in domeinen waar datanauwkeurigheid en -integriteit belangrijke implicaties hebben, zoals financiën, gezondheidszorg en juridische zaken.

Om te voldoen aan de behoefte aan verklaarbaarheid en transparantie moeten zelfherstellende datatechnieken mechanismen bevatten die inzicht geven in het besluitvormingsproces van AI-modellen. Dit kan worden bereikt via verschillende benaderingen:

1. **Gedachtegang**: Door het model te vragen zijn denken "hardop" uit te leggen voordat er wijzigingen in de data worden aangebracht, kan het besluitvormingsproces beter worden begrepen en kunnen er voor mensen leesbare verklaringen worden gegenereerd voor de aangebrachte correcties. De afweging is een iets grotere complexiteit bij het scheiden van de uitleg van de gestructureerde data-output, wat kan worden aangepakt door…
2. **Uitleg Genereren**: Zelfherstellende datawerkers kunnen worden uitgerust met het vermogen om voor mensen leesbare verklaringen te genereren voor de correcties die ze aanbrengen. Dit kan worden bereikt door het model te vragen zijn besluitvormingsproces uit te voeren als gemakkelijk te begrijpen verklaringen die *geïntegreerd zijn in de data zelf*. Een zelfherstellende datawerker zou bijvoorbeeld een rapport kunnen genereren dat de specifieke data-inconsistenties die het heeft geïdentificeerd, de toegepaste correcties en de redenering achter die correcties belicht.
3. **Kenmerkbelangrijkheid**: AI-modellen kunnen worden geïnstrueerd met informatie over het belang van verschillende kenmerken of attributen in het datacorrectieproces als onderdeel van hun richtlijnen. Deze richtlijnen kunnen op hun beurt worden blootgelegd aan menselijke belanghebbenden. Door de belangrijkste factoren te identificeren die de beslissingen van het model beïnvloeden, kunnen belanghebbenden inzicht krijgen in de redenering achter de correcties en hun geldigheid beoordelen.
4. **Logging en Auditing**: Het implementeren van uitgebreide logging- en auditmechanismen is cruciaal voor het handhaven van transparantie in het zelfherstellende dataproces. Elke datacorrectie die door AI-modellen wordt uitgevoerd, moet worden gelogd, inclusief de originele data, de gecorrigeerde data

en de specifieke acties die zijn ondernomen. Dit auditspoor maakt retrospectieve analyse mogelijk en biedt een duidelijk overzicht van de wijzigingen die in de data zijn aangebracht.

5. **Mens-in-de-loop Aanpak**: Het incorporeren van een mens-in-de-loop aanpak kan de verklaarbaarheid en transparantie van zelfherstellende datatechnieken verbeteren. Door menselijke experts te betrekken bij de beoordeling en validatie van door AI gegenereerde correcties, kunnen organisaties ervoor zorgen dat de correcties in lijn zijn met domeinkennis en bedrijfsvereisten. Menselijk toezicht voegt een extra laag van verantwoording toe en maakt het mogelijk om eventuele vooroordelen of fouten in de AI-modellen te identificeren.
6. **Continue Monitoring en Evaluatie**: Regelmatige monitoring en evaluatie van de prestaties van zelfherstellende datatechnieken is essentieel voor het behouden van transparantie en vertrouwen. Door de nauwkeurigheid en effectiviteit van de AI-modellen in de loop van de tijd te beoordelen, kunnen organisaties afwijkingen of anomalieën identificeren en corrigerende maatregelen nemen. Continue monitoring helpt ervoor te zorgen dat het zelfherstellende dataproces betrouwbaar blijft en in lijn is met de gewenste resultaten.

Verklaarbaarheid en transparantie zijn cruciale overwegingen bij het implementeren van zelfherstellende datatechnieken. Door duidelijke verklaringen te geven voor datacorrecties, uitgebreide auditsporen te onderhouden en menselijk toezicht te betrekken, kunnen organisaties vertrouwen opbouwen in het zelfherstellende dataproces en ervoor zorgen dat de wijzigingen in de data te rechtvaardigen zijn en in lijn zijn met bedrijfsdoelstellingen.

Het is belangrijk om een balans te vinden tussen de voordelen van automatisering en de behoefte aan transparantie. Hoewel zelfherstellende datatechnieken de datakwaliteit en efficiëntie aanzienlijk kunnen verbeteren, mag dit niet ten koste gaan van het zicht en de controle over het datacorrectieproces. Door zelfherstellende datawerkers te ontwerpen met verklaarbaarheid en transparantie in gedachten, kunnen organisaties de kracht van

AI benutten terwijl ze het noodzakelijke niveau van verantwoording en vertrouwen in de data behouden.

Onbedoelde Gevolgen

Hoewel zelfherstellende datatechnieken gericht zijn op het verbeteren van datakwaliteit en consistentie, is het cruciaal om bewust te zijn van de mogelijke onbedoelde gevolgen. Geautomatiseerde correcties kunnen, als ze niet zorgvuldig zijn ontworpen en gemonitord, onbedoeld de betekenis of context van de data veranderen, wat kan leiden tot downstream problemen.

Een van de belangrijkste risico's van zelfherstellende data is de introductie van vooroordelen of fouten in het datacorrectieproces. AI-modellen kunnen, net als elk ander softwaresysteem, onderhevig zijn aan vooroordelen die aanwezig zijn in de trainingsdata of die zijn geïntroduceerd door het ontwerp van de algoritmen. Als deze vooroordelen niet worden geïdentificeerd en aangepakt, kunnen ze zich verspreiden via het zelfherstellende dataproces en resulteren in vertekende of onjuiste datawijzigingen.

Neem bijvoorbeeld een zelfherstellende dataverwerker die de taak heeft om inconsistenties in demografische klantgegevens te corrigeren. Als het AI-model vooroordelen heeft geleerd uit historische gegevens, zoals het koppelen van bepaalde beroepen of inkomensniveaus aan specifieke geslachten of etniciteiten, kan het onjuiste aannames maken en de gegevens zodanig wijzigen dat deze vooroordelen worden versterkt. Dit kan leiden tot onnauwkeurige klantprofielen, verkeerde zakelijke beslissingen en mogelijk discriminerende uitkomsten.

Een ander mogelijk onbedoeld gevolg is het verlies van waardevolle informatie of context tijdens het proces van datacorrectie. Zelfherstellende datatechnieken richten zich vaak op het standaardiseren en normaliseren van gegevens om consistentie te waarborgen. In sommige gevallen kan de originele data echter nuances, uitzonderingen of contextuele informatie bevatten die belangrijk zijn voor het begrip van het volledige

beeld. Geautomatiseerde correcties die blindelings standaardisatie afdwingen, kunnen deze waardevolle informatie onbedoeld verwijderen of verhullen.

Stel je bijvoorbeeld een zelfherstellende dataverwerker voor die verantwoordelijk is voor het corrigeren van inconsistenties in medische dossiers. Als de verwerker een medische geschiedenis van een patiënt tegenkomt met een zeldzame aandoening of een ongebruikelijk behandelplan, kan deze proberen de gegevens te normaliseren om ze in een meer gangbaar patroon te laten passen. Hierbij kunnen echter de specifieke details en context verloren gaan die cruciaal zijn voor een accurate weergave van de unieke situatie van de patiënt. Dit verlies aan informatie kan ernstige gevolgen hebben voor de patiëntenzorg en medische besluitvorming.

Om de risico's van onbedoelde gevolgen te beperken, is het essentieel om een proactieve aanpak te hanteren bij het ontwerpen en implementeren van zelfherstellende datatechnieken:

1. **Grondige Tests en Validatie**: Voordat zelfherstellende dataverwerkers in productie worden genomen, is het cruciaal om hun gedrag grondig te testen en te valideren tegen verschillende scenario's. Dit omvat het testen met representatieve datasets die verschillende randgevallen, uitzonderingen en potentiële vooroordelen omvatten. Rigoureus testen helpt bij het identificeren en aanpakken van onbedoelde gevolgen voordat ze impact hebben op echte data.
2. **Continue Monitoring en Evaluatie**: Het implementeren van continue monitoring- en evaluatiemechanismen is essentieel voor het detecteren en beperken van onbedoelde gevolgen in de loop van de tijd. Door regelmatig de uitkomsten van zelfherstellende dataprocessen te beoordelen, de impact op downstream systemen en besluitvorming te analyseren, en feedback van belanghebbenden te verzamelen, kunnen nadelige effecten worden geïdentificeerd en tijdig corrigerende maatregelen worden genomen. Als uw organisatie operationele dashboards heeft, is het waarschijnlijk een goed idee om duidelijk zichtbare metrics toe te voegen die gerelateerd zijn aan

geautomatiseerde datawijzigingen. Het toevoegen van alarmen die gekoppeld zijn aan grote afwijkingen van normale datawijzigingsactiviteit is waarschijnlijk een nog beter idee!

3. **Menselijk Toezicht en Interventie**: Het behouden van menselijk toezicht en de mogelijkheid om in te grijpen in het zelfherstellende dataproces is cruciaal. Hoewel automatisering de efficiëntie aanzienlijk kan verbeteren, is het belangrijk dat menselijke experts de correcties die door AI-modellen worden gemaakt controleren en valideren, vooral in kritieke of gevoelige domeinen. Menselijk oordeel en domeinexpertise kunnen helpen bij het identificeren en aanpakken van eventuele onbedoelde gevolgen.

4. **Verklaarbare AI (XAI) en Transparantie**: Zoals besproken in de vorige subsectie, kan het incorporeren van verklaarbare AI-technieken en het waarborgen van transparantie in het zelfherstellende dataproces helpen bij het beperken van onbedoelde gevolgen. Door duidelijke uitleg te geven over datacorrecties en uitgebreide auditsporen bij te houden, kunnen organisaties de redenering achter de wijzigingen door AI-modellen beter begrijpen en traceren.
5. **Incrementele en Iteratieve Aanpak**: Het adopteren van een incrementele en iteratieve aanpak voor zelfherstellende data kan helpen het risico op onbedoelde gevolgen te minimaliseren. In plaats van geautomatiseerde correcties in één keer op de hele dataset toe te passen, begin je met een subset van de data en breid je de reikwijdte geleidelijk uit naarmate de technieken effectief en betrouwbaar blijken. Dit maakt zorgvuldige monitoring en aanpassing onderweg mogelijk, waardoor de impact van eventuele onbedoelde gevolgen wordt verminderd.
6. **Samenwerking en Feedback**: Het betrekken van belanghebbenden uit verschillende domeinen en het stimuleren van samenwerking en feedback gedurende het zelfherstellende dataproces kan helpen bij het identificeren en aanpakken van onbedoelde gevolgen. Door regelmatig input te vragen van domeinexperts, datagebruikers en eindgebruikers kunnen waardevolle inzichten

worden verkregen in de praktische impact van de datacorrecties en kunnen eventuele over het hoofd geziene problemen worden belicht.

Door proactief de risico's van onbedoelde gevolgen aan te pakken en passende waarborgen te implementeren, kunnen organisaties de voordelen van zelfherstellende datatechnieken benutten en tegelijkertijd potentiële nadelige effecten minimaliseren. Het is belangrijk om zelfherstellende data te benaderen als een iteratief en collaboratief proces, waarbij continue monitoring, evaluatie en verfijning van de technieken plaatsvindt om ervoor te zorgen dat ze in lijn zijn met de gewenste resultaten en de integriteit en betrouwbaarheid van de data behouden blijft.

Bij het overwegen van het gebruik van zelfherstellende datapatronen is het essentieel om deze factoren zorgvuldig te evalueren en de voordelen af te wegen tegen de potentiële risico's en beperkingen. In sommige gevallen kan een hybride aanpak die geautomatiseerde correcties combineert met menselijk toezicht en interventie de meest geschikte oplossing zijn.

Het is ook belangrijk op te merken dat zelfherstellende datatechnieken niet gezien moeten worden als vervanging voor robuuste datavalidatie, invoervalidatie en foutafhandelingsmechanismen. Deze fundamentele praktijken blijven cruciaal voor het waarborgen van data-integriteit en -beveiliging. Zelfherstellende data moet worden gezien als een complementaire aanpak die deze bestaande maatregelen kan aanvullen en verbeteren.

Uiteindelijk hangt de beslissing om zelfherstellende datapatronen toe te passen af van de specifieke vereisten, beperkingen en prioriteiten van uw applicatie. Door zorgvuldig de bovengenoemde overwegingen te beschouwen en deze af te stemmen op de doelen en architectuur van uw applicatie, kunt u weloverwogen beslissingen nemen over wanneer en hoe u zelfherstellende datatechnieken effectief kunt inzetten.

Contextuele Contentgeneratie

Patronen voor Contextuele Contentgeneratie maken gebruik van de kracht van grote taalmodellen (LLMs) om dynamische en contextspecifieke content binnen applicaties te genereren. Deze categorie patronen erkent het belang van het leveren van gepersonaliseerde en relevante content aan gebruikers, gebaseerd op hun specifieke behoeften, voorkeuren en zelfs eerdere en huidige interacties met de applicatie.

> In de context van deze benadering verwijst “content” zowel naar primaire content (zoals blogposts, artikelen, etc.) als naar meta-content, zoals aanbevelingen voor primaire content.

Patronen voor Contextuele Contentgeneratie kunnen een cruciale rol spelen bij het verbeteren van uw gebruikersbetrokkenheidsniveaus, het bieden van op maat

gemaakte ervaringen, en het automatiseren van contentcreatietaken voor zowel u als uw gebruikers. Door de patronen die we in dit hoofdstuk beschrijven te gebruiken, kunt u applicaties creëren die dynamisch content genereren en zich in realtime aanpassen aan context en input.

De patronen werken door LLMs te integreren in de output van de applicatie, variërend van de gebruikersinterface (soms aangeduid als "chrome"), tot e-mails en andere vormen van notificaties, evenals alle contentgeneratiepijplijnen.

Wanneer een gebruiker met de applicatie interacteert of een specifiek contentverzoek activeert, legt de applicatie de relevante context vast, zoals gebruikersvoorkeuren, eerdere interacties of specifieke prompts. Deze contextuele informatie wordt vervolgens samen met eventueel benodigde sjablonen of richtlijnen in het LLM ingevoerd en gebruikt om tekstuele output te produceren die anders hardgecodeerd, in een database opgeslagen of algoritmisch gegenereerd zou moeten worden.

De door LLM gegenereerde content kan verschillende vormen aannemen, zoals gepersonaliseerde aanbevelingen, dynamische productbeschrijvingen, aangepaste e-mailreacties, of zelfs complete artikelen of blogposts. Een van de meest radicale toepassingen van deze content die ik meer dan een jaar geleden introduceerde, is het dynamisch genereren van UI-elementen zoals formulierlabels, tooltips en andere vormen van verklarende tekst.

Personalisatie

Een van de belangrijkste voordelen van Contextuele Contentgeneratiepatronen is het vermogen om zeer gepersonaliseerde ervaringen aan gebruikers te leveren. Door content te genereren op basis van gebruikersspecifieke context, stellen deze patronen applicaties in staat om content af te stemmen op de individuele interesses, voorkeuren en interacties van gebruikers.

Personalisatie gaat verder dan simpelweg de naam van een gebruiker in generieke

content invoegen. Het omvat het benutten van de rijke context die beschikbaar is over elke gebruiker om content te genereren die resoneert met hun specifieke behoeften en wensen. Deze context kan een breed scala aan factoren omvatten, zoals:

1. **Gebruikersprofielinformatie**: Op het meest algemene niveau van toepassing van deze techniek kunnen demografische gegevens, interesses, voorkeuren en andere profielkenmerken worden gebruikt om content te genereren die aansluit bij de achtergrond en eigenschappen van de gebruiker.
2. **Gedragsgegevens**: De eerdere interacties van een gebruiker met de applicatie, zoals bekeken pagina's, aangeklikte links of gekochte producten, kunnen waardevolle inzichten bieden in hun gedrag en interesses. Deze gegevens kunnen worden gebruikt om contentvoorstellen te genereren die hun betrokkenheidspatronen weerspiegelen en hun toekomstige behoeften voorspellen.
3. **Contextuele Factoren**: De huidige context van de gebruiker, zoals hun locatie, apparaat, tijd van de dag, of zelfs het weer, kan het contentgeneratieproces beïnvloeden. Een reisapplicatie zou bijvoorbeeld een AI-worker kunnen hebben die gepersonaliseerde aanbevelingen kan genereren op basis van de huidige locatie van de gebruiker en de heersende weersomstandigheden.

Door deze contextuele factoren te benutten, stellen Contextuele Contentgeneratiepatronen applicaties in staat om content te leveren die op maat gemaakt lijkt voor elke individuele gebruiker. Dit niveau van personalisatie heeft verschillende belangrijke voordelen:

1. **Verhoogde Betrokkenheid**: Gepersonaliseerde content trekt de aandacht van gebruikers en houdt ze betrokken bij de applicatie. Wanneer gebruikers het gevoel hebben dat de content relevant is en direct inspeelt op hun behoeften, zijn ze eerder geneigd meer tijd te besteden aan het interacteren met de applicatie en het verkennen van de functies.

2. **Verbeterde Gebruikerstevredenheid**: Gepersonaliseerde content laat zien dat de applicatie de unieke vereisten van de gebruiker begrijpt en daarom geeft. Door content te bieden die behulpzaam, informatief en afgestemd is op hun interesses, kan de applicatie de gebruikerstevredenheid vergroten en een sterkere band met zijn gebruikers opbouwen.
3. **Hogere Conversieratio's**: In de context van e-commerce of marketingapplicaties kan gepersonaliseerde content een significante impact hebben op conversieratio's. Door gebruikers producten, aanbiedingen of aanbevelingen te presenteren die zijn afgestemd op hun voorkeuren en gedrag, kan de applicatie de kans vergroten dat gebruikers gewenste acties ondernemen, zoals het doen van een aankoop of het aanmelden voor een dienst.

Productiviteit

Contextuele Contentgeneratiepatronen kunnen bepaalde vormen van productiviteit aanzienlijk verhogen door de behoefte aan handmatige contentgeneratie en bewerking in creatieve processen te verminderen. Door gebruik te maken van de kracht van LLMs kunt u op grote schaal hoogwaardige content genereren, waardoor u tijd en moeite bespaart die uw contentmakers en ontwikkelaars anders zouden moeten besteden aan vervelend handmatig werk.

Traditioneel moeten contentmakers onderzoek doen, schrijven, redigeren en content formatteren om ervoor te zorgen dat deze voldoet aan de vereisten van de applicatie en de verwachtingen van gebruikers. Dit proces kan tijdrovend en arbeidsintensief zijn, vooral naarmate de hoeveelheid content groeit.

Met Contextuele Contentgeneratiepatronen kan het contentcreatieproces echter grotendeels worden geautomatiseerd. LLMs kunnen samenhangende, grammaticaal correcte en contextueel relevante content genereren op basis van de gegeven prompts en richtlijnen. Deze automatisering biedt verschillende productiviteitsvoordelen:

1. **Verminderde Handmatige Inspanning**: Door contentageneratietaken te delegeren aan LLMs kunnen contentmakers zich richten op taken van hoger niveau, zoals contentstrategie, ideeontwikkeling en kwaliteitsborging. Ze kunnen de nodige context, sjablonen en richtlijnen aan het LLM verstrekken en het de daadwerkelijke contentgeneratie laten afhandelen. Dit vermindert de handmatige inspanning die nodig is voor schrijven en redigeren, waardoor contentmakers productiever en efficiënter kunnen werken.
2. **Snellere Contentcreatie**: LLMs kunnen veel sneller content genereren dan menselijke schrijvers. Met de juiste prompts en richtlijnen kan een LLM meerdere stukken content produceren in enkele seconden of minuten. Deze snelheid stelt applicaties in staat om in een veel hoger tempo content te genereren, waardoor ze kunnen bijblijven met de eisen van gebruikers en het steeds veranderende digitale landschap.

> Leidt snellere contentcreatie tot een "tragedy of the commons" situatie waarbij het internet overspoeld raakt met content die niemand leest? Helaas vermoed ik dat het antwoord ja is.

3. **Consistentie en Kwaliteit**: LLMs kunnen moeiteloos content herzien zodat deze consistent is in stijl, toon en kwaliteit. Met duidelijke richtlijnen en voorbeelden kunnen bepaalde soorten applicaties (zoals nieuwsredacties, PR, etc.) ervoor zorgen dat hun door mensen gegenereerde content aansluit bij hun merkidentiteit en voldoet aan de gewenste kwaliteitsnormen. Deze consistentie vermindert de behoefte aan uitgebreide bewerking en herzieningen, wat tijd en moeite bespaart in het contentcreatieproces.
4. **Iteratie en Optimalisatie**: Contextuele Contentgeneratiepatronen maken snelle iteratie en optimalisatie van content mogelijk. Door het aanpassen van de

prompts, sjablonen of richtlijnen die aan het LLM worden verstrekt, kunnen uw applicaties snel contentvariaties genereren en verschillende benaderingen testen op een geautomatiseerde manier die in het verleden nooit mogelijk was. Dit iteratieve proces maakt sneller experimenteren en verfijnen van contentstrategieën mogelijk, wat in de loop van de tijd leidt tot effectievere en meer betrokken content. Deze specifieke techniek kan een absolute game-changer zijn voor applicaties zoals e-commerce die leven en sterven op basis van bouncepercentages en betrokkenheid

Het is belangrijk op te merken dat hoewel Contextuele Contentgeneratiepatronen de productiviteit aanzienlijk kunnen verbeteren, ze de noodzaak van menselijke betrokkenheid niet volledig wegnemen. Contentmakers en redacteuren spelen nog steeds een cruciale rol bij het bepalen van de algemene contentstrategie, het geven van sturing aan het LLM en het waarborgen van de kwaliteit en geschiktheid van de gegenereerde content.

Door de meer repetitieve en tijdrovende aspecten van contentcreatie te automatiseren, maken Contextuele Contentgeneratiepatronen waardevolle menselijke tijd en middelen vrij die kunnen worden ingezet voor taken met een hogere waarde. Deze verhoogde productiviteit stelt u in staat om meer gepersonaliseerde en betrokken content aan gebruikers te leveren terwijl contentcreatieprocessen worden geoptimaliseerd.

Snelle Iteratie en Experimentatie

Contextuele Contentgeneratiepatronen stellen u in staat om snel te itereren en te experimenteren met verschillende contentvariaties, waardoor snellere optimalisatie en verfijning van uw contentstrategie mogelijk wordt. U kunt in enkele seconden meerdere versies van content genereren, simpelweg door de context, sjablonen of richtlijnen die aan het model worden verstrekt aan te passen.

Deze snelle iteratiemogelijkheid biedt verschillende belangrijke voordelen:

1. **Testen en Optimalisatie**: Met de mogelijkheid om snel contentvariaties te genereren, kunt u eenvoudig verschillende benaderingen testen en hun effectiviteit meten. U kunt bijvoorbeeld meerdere versies van een productbeschrijving of marketingboodschap genereren, elk afgestemd op een specifiek gebruikerssegment of context. Door gebruikersbetrokkenheidsmetrieken te analyseren, zoals doorklikratio's of conversiepercentages, kunt u de meest effectieve contentvariaties identificeren en uw contentstrategie dienovereenkomstig optimaliseren.

2. **A/B-testen**: Contextuele Contentgeneratiepatronen maken naadloos A/B-testen van content mogelijk. U kunt twee of meer variaties van content genereren en deze willekeurig aan verschillende gebruikersgroepen tonen. Door de prestaties van elke variatie te vergelijken, kunt u bepalen welke content het beste resoneert met uw doelgroep. Deze datagestuurde aanpak stelt u in staat om geïnformeerde beslissingen te nemen en uw content voortdurend te verfijnen om gebruikersbetrokkenheid te maximaliseren en uw gewenste resultaten te bereiken.
3. **Personalisatie-experimenten**: Snelle iteratie en experimentatie zijn bijzonder waardevol als het gaat om personalisatie. Met Contextuele Contentgeneratiepatronen kunt u snel gepersonaliseerde contentvariaties genereren op basis van verschillende gebruikerssegmenten, voorkeuren of gedragingen. Door te experimenteren met verschillende personalisatiestrategieën kunt u de meest effectieve benaderingen identificeren voor het betrekken van individuele gebruikers en het leveren van op maat gemaakte ervaringen.
4. **Aanpassen aan Veranderende Trends**: Het vermogen om snel te itereren en te experimenteren stelt je in staat om flexibel te blijven en je aan te passen aan veranderende trends en gebruikersvoorkeuren. Wanneer nieuwe onderwerpen, zoekwoorden of gebruikersgedrag ontstaan, kun je snel content genereren die

aansluit bij deze trends. Door voortdurend te experimenteren en je content te verfijnen, kun je relevant blijven en een concurrentievoordeel behouden in het steeds veranderende digitale landschap.

5. **Kosteneffectief Experimenteren**: Traditioneel content-experimenteren brengt vaak aanzienlijke tijd en middelen met zich mee, omdat contentmakers handmatig verschillende variaties moeten ontwikkelen en testen. Met Contextuele Content Generatie-patronen worden de kosten van experimenteren echter sterk verminderd. Grote taalmodellen kunnen snel en op schaal contentvariaties genereren, waardoor je een breed scala aan ideeën en benaderingen kunt verkennen zonder substantiële kosten.

Om het maximale uit snelle iteratie en experimenten te halen, is het belangrijk om een goed gedefinieerd experimenteerframework te hebben. Dit framework moet het volgende bevatten:

- Duidelijke doelstellingen en hypotheses voor elk experiment
- Geschikte meetwaarden en trackingmechanismen om contentprestaties te meten
- Segmentatie- en targetingstrategieën om ervoor te zorgen dat relevante contentvariaties bij de juiste gebruikers terechtkomen
- Analyse- en rapportagetools om inzichten te verkrijgen uit de experimentele data
- Een proces voor het integreren van leerpunten en optimalisaties in je contentstrategie

Door snelle iteratie en experimenteren te omarmen, kun je je content voortdurend verfijnen en optimaliseren, zodat deze boeiend, relevant en effectief blijft in het bereiken van de doelen van je applicatie. Deze flexibele benadering van contentcreatie stelt je in staat om voorop te blijven lopen en uitzonderlijke gebruikerservaringen te leveren.

Schaalbaarheid en Efficiëntie

Naarmate applicaties groeien en de vraag naar gepersonaliseerde content toeneemt, maken contextuele content generatie-patronen een efficiënte opschaling van

contentcreatie mogelijk. Grote taalmodellen kunnen gelijktijdig content genereren voor een groot aantal gebruikers en contexten, zonder dat er een evenredige toename van menselijke middelen nodig is. Deze schaalbaarheid stelt applicaties in staat om gepersonaliseerde ervaringen te leveren aan een groeiend gebruikersbestand zonder hun contentcreatiecapaciteiten te overbelasten.

Merk op dat contextuele contentgeneratie effectief kan worden gebruikt om je applicatie "on the fly" te internationaliseren. Dat is precies wat ik heb gedaan met mijn Instant18n Gem om Olympia in meer dan een half dozijn talen aan te bieden, ook al zijn we nog geen jaar oud.

AI-Aangedreven Lokalisatie

Als je me even toestaat op te scheppen, denk ik dat mijn Instant18n-bibliotheek voor Rails-apps een baanbrekend voorbeeld is van het "Contextuele Content Generatie"-patroon in actie, dat het transformatieve potentieel van AI in applicatieontwikkeling laat zien. Deze gem maakt gebruik van de kracht van OpenAI's GPT grote-taalmodel om de manier waarop internationalisering en lokalisatie worden behandeld in Rails-applicaties te revolutioneren.

Traditioneel gezien omvat het internationaliseren van een Rails-applicatie het handmatig definiëren van vertaalsleutels en het leveren van bijbehorende vertalingen voor elke ondersteunde taal. Dit proces kan tijdrovend en arbeidsintensief zijn en gevoelig voor inconsistenties. Met de Instant18n-gem wordt het paradigma van lokalisatie echter volledig opnieuw gedefinieerd.

Door de integratie van een groot taalmodel stelt de Instant18n-gem je in staat om vertalingen on-the-fly te genereren, gebaseerd op de context en betekenis van de tekst. In plaats van te vertrouwen op voorgedefinieerde vertaalsleutels en statische vertalingen, vertaalt de gem dynamisch tekst met behulp van de kracht van AI. Deze aanpak biedt verschillende belangrijke voordelen:

1. **Naadloze Lokalisatie**: Met de Instant18n-gem hoeven ontwikkelaars niet langer handmatig vertaalbestanden te definiëren en te onderhouden voor elke ondersteunde taal. De gem genereert automatisch vertalingen op basis van de aangeleverde tekst en de gewenste doeltaal, waardoor het lokalisatieproces moeiteloos en naadloos verloopt.
2. **Contextuele Nauwkeurigheid**: AI kan voldoende context krijgen om de nuances van de te vertalen tekst te begrijpen. Het kan rekening houden met de omringende context, idiomen en culturele verwijzingen om vertalingen te genereren die accuraat, natuurlijk klinkend en contextueel passend zijn.
3. **Uitgebreide Taalondersteuning**: De Instant18n-gem maakt gebruik van de uitgebreide kennis en taalkundige mogelijkheden van GPT, waardoor vertalingen naar een uitgebreid scala aan talen mogelijk zijn. Van veelvoorkomende talen zoals Spaans en Frans tot meer obscure of fictieve talen zoals Klingon en Elfs, de gem kan een breed scala aan vertaalvereisten aan.
4. **Flexibiliteit en Creativiteit**: De gem gaat verder dan traditionele taalvertalingen en maakt creatieve en onconventionele lokalisatieopties mogelijk. Ontwikkelaars kunnen tekst vertalen naar verschillende stijlen, dialecten of zelfs fictieve talen, wat nieuwe mogelijkheden opent voor unieke gebruikerservaringen en boeiende content.
5. **Prestatie-optimalisatie**: De Instant18n-gem bevat cachingmechanismen om de prestaties te verbeteren en de overhead van herhaalde vertalingen te verminderen. Vertaalde tekst wordt gecached, waardoor volgende verzoeken voor dezelfde vertaling snel kunnen worden geleverd zonder de noodzaak van redundante API-aanroepen.

De Instant18n-gem illustreert de kracht van het "Contextuele Content Generatie"-patroon door AI te gebruiken om dynamisch gelokaliseerde content te genereren. Het laat zien hoe AI kan worden geïntegreerd in de kernfunctionaliteit van een Rails-applicatie, waardoor de manier waarop ontwikkelaars internationalisering en lokalisatie benaderen wordt getransformeerd.

Door de noodzaak van handmatig vertalingsbeheer weg te nemen en directe vertalingen op basis van context mogelijk te maken, bespaart de Instant18n gem ontwikkelaars aanzienlijk veel tijd en moeite. Het stelt hen in staat zich te concentreren op het bouwen van de kernfunctionaliteiten van hun applicatie, terwijl ze er zeker van kunnen zijn dat het lokalisatie-aspect naadloos en accuraat wordt afgehandeld.

Het Belang van Gebruikerstests en Feedback

Tot slot, verlies nooit het belang van gebruikerstests en feedback uit het oog. Het is cruciaal om te valideren dat contextuele contentgeneratie aan de verwachtingen van gebruikers voldoet en in lijn is met de doelen van de applicatie. Blijf gegenereerde content continu itereren en verfijnen op basis van gebruikersinzichten en analyses. Als je dynamische content genereert op een schaal die onmogelijk handmatig te valideren is door jou en je team, overweeg dan om feedbackmechanismen toe te voegen waarmee gebruikers vreemde of onjuiste content kunnen melden, samen met een uitleg waarom. Die waardevolle feedback kan zelfs worden doorgespeeld aan een AI-worker die de taak heeft om aanpassingen te maken aan de component die de content heeft gegenereerd!

Generatieve UI

Aandacht is tegenwoordig zo kostbaar dat effectieve gebruikersbetrokkenheid nu vraagt om software-ervaringen die niet alleen naadloos en intuïtief zijn, maar ook sterk gepersonaliseerd zijn op individuele behoeften, voorkeuren en contexten. Als gevolg hiervan staan ontwerpers en ontwikkelaars steeds vaker voor de uitdaging om gebruikersinterfaces te creëren die zich kunnen aanpassen aan de unieke vereisten van elke gebruiker *op schaal*.

Generatieve UI (GenUI) is een werkelijk revolutionaire benadering van gebruikersinterface-ontwerp die gebruik maakt van de kracht van grote taalmodellen (LLMs) om zeer gepersonaliseerde en dynamische gebruikerservaringen on-the-fly te creëren. Ik wilde je in dit boek in ieder geval een introductie geven over GenUI, omdat ik geloof dat het een van de groenste groene weide kansen is die momenteel bestaat op het gebied van applicatieontwerp en frameworks. Ik ben ervan overtuigd dat er tientallen of meer nieuwe succesvolle commerciële en open-source projecten in deze

specifieke niche zullen ontstaan.

In de kern combineert GenUI de principes van Contextuele Contentgeneratie met geavanceerde AI-technieken om gebruikersinterface-elementen zoals tekst, afbeeldingen en layouts dynamisch te genereren, gebaseerd op een diep begrip van de context, voorkeuren en doelen van de gebruiker. GenUI stelt ontwerpers en ontwikkelaars in staat om interfaces te creëren die zich aanpassen en ontwikkelen in reactie op gebruikersinteracties, waardoor een niveau van personalisatie mogelijk wordt dat voorheen onbereikbaar was.

GenUI vertegenwoordigt een fundamentele verandering in de manier waarop we gebruikersinterface-ontwerp benaderen. In plaats van te ontwerpen voor de massa, stelt GenUI ons in staat om voor het individu te ontwerpen. Gepersonaliseerde content en interfaces hebben de potentie om gebruikerservaringen te creëren die bij elke gebruiker op een dieper niveau resoneren, waardoor betrokkenheid, tevredenheid en loyaliteit toenemen.

Als een ultramoderne techniek zit de overgang naar GenUI vol met conceptuele en praktische uitdagingen. Het integreren van AI in het ontwerpproces, ervoor zorgen dat de gegenereerde interfaces niet alleen gepersonaliseerd zijn maar ook bruikbaar, toegankelijk en in lijn met de algemene merkbeleving en gebruikerservaring, dit zijn allemaal uitdagingen die GenUI een bezigheid maken voor de weinigen, niet de velen. Bovendien roept de betrokkenheid van AI vragen op over gegevensprivacy, transparantie en mogelijk zelfs ethische implicaties.

Ondanks de uitdagingen hebben gepersonaliseerde ervaringen op schaal de kracht om de manier waarop we met digitale producten en diensten omgaan volledig te transformeren. Het opent mogelijkheden voor het creëren van inclusieve en toegankelijke interfaces die tegemoetkomen aan de diverse behoeften van gebruikers, ongeacht hun mogelijkheden, achtergrond of voorkeuren.

In dit hoofdstuk zullen we het concept van GenUI verkennen, waarbij we enkele bepalende kenmerken, belangrijke voordelen en potentiële uitdagingen onderzoeken.

We beginnen met het beschouwen van de meest basale en toegankelijke vorm van GenUI: het genereren van tekstkopij voor anderszins traditioneel ontworpen en geïmplementeerde gebruikersinterfaces.

Het Genereren van Kopij voor Gebruikersinterfaces

Tekstelementen die bestaan in de chrome van je applicatie, zoals formulierlabels, tooltips en verklarende tekst, zijn meestal hardgecodeerd in de templates of UI-componenten, wat zorgt voor een consistente maar generieke ervaring voor alle gebruikers. Door gebruik te maken van contextuele contentgeneratiepatronen, kun je deze statische elementen transformeren in dynamische, contextbewuste en gepersonaliseerde componenten.

Gepersonaliseerde Formulieren

Formulieren zijn een alomtegenwoordig onderdeel van web- en mobiele applicaties en dienen als het primaire middel voor het verzamelen van gebruikersinvoer. Traditionele formulieren bieden echter vaak een generieke en onpersoonlijke ervaring, met standaardlabels en velden die niet altijd aansluiten bij de specifieke context of behoeften van de gebruiker. Gebruikers zijn eerder geneigd formulieren in te vullen die afgestemd zijn op hun behoeften en voorkeuren, wat leidt tot hogere conversiepercentages en gebruikerstevredenheid.

Het is echter belangrijk om een balans te vinden tussen personalisatie en consistentie. Hoewel het aanpassen van formulieren aan individuele gebruikers gunstig kan zijn, is het cruciaal om een niveau van vertrouwdheid en voorspelbaarheid te behouden. Gebruikers moeten formulieren nog steeds gemakkelijk kunnen herkennen en navigeren, zelfs met gepersonaliseerde elementen.

Hier zijn enkele gepersonaliseerde formulierideeën ter inspiratie:

Contextuele Veldsuggesties

GenUI kan de eerdere interacties, voorkeuren en gegevens van de gebruiker analyseren om intelligente veldsuggesties als voorspellingen te bieden. Als de gebruiker bijvoorbeeld eerder hun verzendadres heeft ingevoerd, kan het formulier automatisch de relevante velden invullen met hun opgeslagen informatie. Dit bespaart niet alleen tijd, maar laat ook zien dat de applicatie de voorkeuren van de gebruiker begrijpt en onthoudt.

Wacht even, is deze techniek niet iets dat ook zonder AI gedaan zou kunnen worden? Natuurlijk, maar de schoonheid van het aandrijven van dit soort functionaliteit met AI is tweeledig: 1) hoe eenvoudig het te implementeren is en 2) hoe flexibel het blijft terwijl je UI verandert en evolueert in de loop van de tijd.

Laten we een service opzetten voor ons theoretische orderafhandelingssysteem, die proactief probeert het juiste verzendadres voor de gebruiker in te vullen.

```
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
```

Dit voorbeeld is sterk vereenvoudigd, maar zou in de meeste gevallen moeten werken. Het idee is om de AI een inschatting te laten maken op dezelfde manier als een mens dat zou doen. Om duidelijk te maken waar ik het over heb, laten we eens naar wat voorbeeldgegevens kijken:

```
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"}
```

```
]
```

Heb je het patroon in de data opgemerkt? Ik garandeer je dat dit eenvoudig werk is voor een LLM. Om dit te demonstreren, laten we aan GPT-4 vragen wat het meest waarschijnlijke verzendadres is voor een “thermometer”.

```
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.
```

Als je bij jezelf denkt dat het overdreven is om een duur model zoals GPT-4 voor deze taak te gebruiken, heb je gelijk! Ik probeerde dezelfde prompt op Mistral 7B Instruct en het produceerde het volgende antwoord met 75 tokens per seconde, tegen een bescheiden kostprijs van $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.
```

Is de overhead en kosten van deze techniek het waard om een afrekenervaring magischer te maken? Voor veel online retailers, absoluut. En zoals het eruitziet, zullen de kosten van AI-berekeningen alleen maar dalen, vooral voor reguliere open source model hosting providers in een race naar de bodem.

Gebruik een Prompt Template en StructuredIO samen met Response Fencing om dit soort chatvervolmaking te optimaliseren.

Adaptieve veldvolgorde

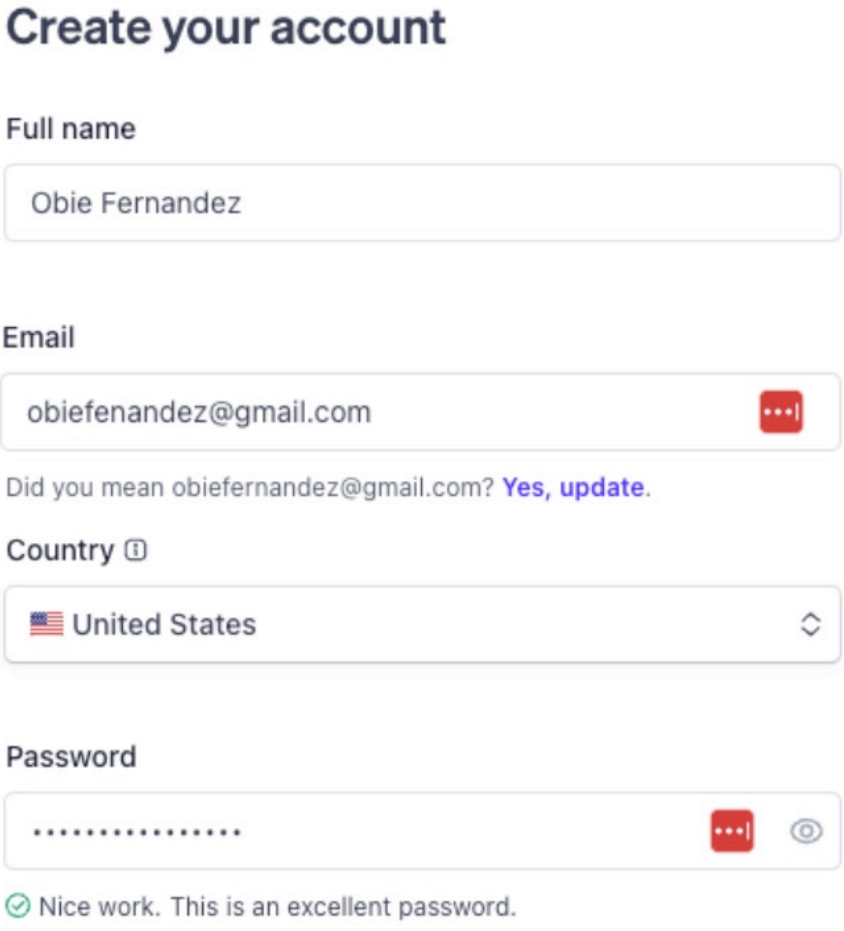
De volgorde waarin formuliervelden worden gepresenteerd kan een aanzienlijke invloed hebben op de gebruikerservaring en voltooiingspercentages. Met GenUI kun je de veldvolgorde dynamisch aanpassen op basis van de context van de gebruiker en het belang van elk veld. Als een gebruiker bijvoorbeeld een registratieformulier invult voor een fitness-app, kan het formulier prioriteit geven aan velden die gerelateerd zijn aan hun fitnessdoelen en voorkeuren, waardoor het proces relevanter en aantrekkelijker wordt.

Gepersonaliseerde microtekst

De instructietekst, foutmeldingen en andere microtekst die bij formulieren horen kunnen ook worden gepersonaliseerd met behulp van GenUI. In plaats van algemene foutmeldingen zoals “Ongeldig e-mailadres” weer te geven, kun je meer behulpzame en contextuele berichten genereren zoals “Voer een geldig e-mailadres in om je orderbevestiging te ontvangen.” Deze gepersonaliseerde details kunnen de formulierervaring gebruiksvriendelijker en minder frustrerend maken.

Gepersonaliseerde validatie

In dezelfde lijn als Gepersonaliseerde Microtekst, zou je AI kunnen gebruiken om het formulier te valideren op manieren die magisch lijken. Stel je voor dat je een AI een gebruikersprofielformulier laat valideren, waarbij wordt gezocht naar mogelijke fouten op *semantisch* niveau.



Figuur 9. Kun je de semantische validatie ontdekken?

Progressieve onthulling

GenUI kan intelligent bepalen welke formuliervelden essentieel zijn op basis van de context van de gebruiker en geleidelijk aanvullende velden onthullen wanneer nodig. Deze progressieve onthulling techniek helpt de cognitieve belasting te verminderen en maakt het invullen van formulieren beter beheerbaar. Als een gebruiker zich

bijvoorbeeld aanmeldt voor een basisabonnement, kan het formulier aanvankelijk alleen de essentiële velden tonen, en naarmate de gebruiker vordert of specifieke opties selecteert, kunnen aanvullende relevante velden dynamisch worden geïntroduceerd.

Contextbewuste verklarende tekst

Tooltips worden vaak gebruikt om aanvullende informatie of begeleiding te bieden aan gebruikers wanneer ze over specifieke elementen bewegen of ermee interacteren. Met een "Contextuele Content Generatie" aanpak kun je tooltips genereren die zich aanpassen aan de context van de gebruiker en relevante informatie bieden. Als een gebruiker bijvoorbeeld een complexe functie verkent, kan de tooltip gepersonaliseerde tips of voorbeelden bieden op basis van hun eerdere interacties of vaardigheidsniveau.

Verklarende tekst, zoals instructies, beschrijvingen of hulpberichten, kan dynamisch worden gegenereerd op basis van de context van de gebruiker. In plaats van algemene uitleg te presenteren, kun je LLMs gebruiken om tekst te genereren die is afgestemd op de specifieke behoeften of vragen van de gebruiker. Als een gebruiker bijvoorbeeld moeite heeft met een bepaalde stap in een proces, kan de verklarende tekst gepersonaliseerde begeleiding of probleemoplossingstips bieden.

Microtekst verwijst naar de kleine stukjes tekst die gebruikers door je applicatie leiden, zoals knoplabels, foutmeldingen of bevestigingsprompts. Door de Contextuele Content Generatie aanpak toe te passen op microtekst, kun je een adaptieve UI creëren die reageert op de acties van de gebruiker en relevante en behulpzame tekst biedt. Als een gebruiker bijvoorbeeld op het punt staat een kritieke actie uit te voeren, kan de bevestigingsprompt dynamisch worden gegenereerd om een duidelijke en gepersonaliseerde boodschap te geven.

Gepersonaliseerde verklarende tekst en tooltips kunnen het onboardingproces voor nieuwe gebruikers aanzienlijk verbeteren. Door contextspecifieke begeleiding en voorbeelden te bieden, kun je gebruikers helpen de applicatie snel te begrijpen en te navigeren, waardoor de leercurve wordt verkort en de adoptie wordt verhoogd.

Dynamische en contextbewuste chrome-elementen kunnen de applicatie ook intuïtiever en aantrekkelijker maken. Gebruikers zijn eerder geneigd om met functies te interacteren en deze te verkennen wanneer de bijbehorende tekst is afgestemd op hun specifieke behoeften en interesses.

Tot nu toe hebben we ideeën besproken voor het verbeteren van bestaande UI-paradigma's met AI, maar hoe zit het met het radicaal heroverwegen van hoe gebruikersinterfaces worden ontworpen en geïmplementeerd?

Definitie van Generatieve UI

In tegenstelling tot traditioneel UI-ontwerp, waarbij ontwerpers vaste, statische interfaces creëren, hint GenUI naar een toekomst waarin onze software beschikt over flexibele, gepersonaliseerde ervaringen die in realtime kunnen evolueren en zich aanpassen. Elke keer dat we een AI-gestuurde conversatie-interface gebruiken, laten we de AI zich aanpassen aan de specifieke behoeften van de gebruiker. GenUI gaat een stap verder door dat niveau van aanpasbaarheid toe te passen op de *visuele* interface van software.

De reden dat het mogelijk is om vandaag de dag te experimenteren met GenUI-ideeën is dat grote taalmodellen al programmeren begrijpen en hun basiskennis UI-technologieën en frameworks omvat. De vraag is dus of grote taalmodellen kunnen worden gebruikt om UI-elementen te genereren, zoals tekst, afbeeldingen, layouts en zelfs complete interfaces, die zijn toegesneden op elke individuele gebruiker. Het model zou kunnen worden geïnstrueerd om rekening te houden met verschillende factoren, zoals eerdere interacties van de gebruiker, aangegeven voorkeuren, demografische informatie en de huidige gebruikscontext, om zeer gepersonaliseerde en relevante interfaces te creëren.

GenUI verschilt op verschillende belangrijke manieren van traditioneel gebruikersinterface-ontwerp:

1. **Dynamisch en Adaptief**: Traditioneel UI-ontwerp omvat het creëren van vaste, statische interfaces die voor alle gebruikers hetzelfde blijven. GenUI daarentegen maakt interfaces mogelijk die zich dynamisch kunnen aanpassen en veranderen op basis van gebruikersbehoeften en context. Dit betekent dat dezelfde applicatie verschillende interfaces kan presenteren aan verschillende gebruikers of zelfs aan dezelfde gebruiker in verschillende situaties.
2. **Personalisatie op Schaal**: Bij traditioneel ontwerp is het vaak onpraktisch om gepersonaliseerde ervaringen voor elke gebruiker te creëren vanwege de benodigde tijd en middelen. GenUI maakt daarentegen personalisatie op schaal mogelijk. Door gebruik te maken van AI kunnen ontwerpers interfaces creëren die zich automatisch aanpassen aan de unieke behoeften en voorkeuren van elke gebruiker, zonder handmatig aparte interfaces te hoeven ontwerpen en ontwikkelen voor elk gebruikerssegment.
3. **Focus op Resultaten**: Traditioneel UI-ontwerp richt zich vaak op het creëren van visueel aantrekkelijke en functionele interfaces. Hoewel deze aspecten nog steeds belangrijk zijn bij GenUI, verschuift de primaire focus naar het bereiken van gewenste gebruikersresultaten. GenUI streeft ernaar interfaces te creëren die geoptimaliseerd zijn voor de specifieke doelen en taken van elke gebruiker, waarbij bruikbaarheid en effectiviteit voorrang krijgen boven puur esthetische overwegingen.
4. **Continue Leren en Verbeteren**: GenUI-systemen kunnen continu leren en verbeteren op basis van gebruikersinteracties en feedback. Terwijl gebruikers werken met de gegenereerde interfaces, kunnen de AI-modellen data verzamelen over gebruikersgedrag, voorkeuren en resultaten, en deze informatie gebruiken om toekomstige interface-generaties te verfijnen en optimaliseren. Dit iteratieve leerproces stelt GenUI-systemen in staat om in de loop van de tijd steeds effectiever te worden in het voldoen aan gebruikersbehoeften.

Het is belangrijk op te merken dat GenUI niet hetzelfde is als AI-ondersteunde ontwerptools, zoals tools die suggesties geven of bepaalde ontwerptaken automatiseren.

Hoewel deze tools nuttig kunnen zijn bij het stroomlijnen van het ontwerpproces, zijn ze nog steeds afhankelijk van ontwerpers om eindbesluiten te nemen en statische interfaces te creëren. Bij GenUI daarentegen speelt het AI-systeem een actievere rol in de daadwerkelijke generatie en aanpassing van interfaces op basis van gebruikersdata en context.

GenUI vertegenwoordigt een belangrijke verschuiving in hoe we gebruikersinterface-ontwerp benaderen, waarbij we ons verwijderen van one-size-fits-all oplossingen en bewegen naar zeer gepersonaliseerde, adaptieve ervaringen. Door gebruik te maken van de kracht van AI heeft GenUI het potentieel om de manier waarop we interacteren met digitale producten en diensten te revolutioneren, door interfaces te creëren die intuïtiever, boeiender en effectiever zijn voor elke individuele gebruiker.

Voorbeeld

Om het concept van GenUI te illustreren, laten we een hypothetische fitness-applicatie genaamd “FitAI” bekijken. Deze app heeft als doel gepersonaliseerde trainingsschema’s en voedingsadvies te bieden aan gebruikers op basis van hun individuele doelen, fitnessniveaus en voorkeuren.

In een traditionele UI-ontwerpbenadering zou FitAI een vaste set schermen en elementen kunnen hebben die voor alle gebruikers hetzelfde zijn. Met GenUI zou de interface van de app zich echter dynamisch kunnen aanpassen aan de unieke behoeften en context van elke gebruiker.

Deze aanpak is nogal vergezocht om in 2024 te implementeren en heeft mogelijk niet eens voldoende ROI, maar het is mogelijk.

Zo zou het kunnen werken:

1. **Onboarding**:

- In plaats van een standaard vragenlijst gebruikt FitAI een conversationele AI om informatie te verzamelen over de doelen, het huidige fitnessniveau en de voorkeuren van de gebruiker.
- Op basis van deze eerste interactie genereert de AI een gepersonaliseerde dashboard-layout, waarbij de functies en informatie die het meest relevant zijn voor de doelen van de gebruiker worden benadrukt.
- Huidige AI-technologie zou kunnen beschikken over een selectie van schermcomponenten om te gebruiken bij het samenstellen van het gepersonaliseerde dashboard.
- Toekomstige AI-technologie zou de rol van een ervaren UI-ontwerper kunnen overnemen en het dashboard daadwerkelijk *vanaf nul* kunnen creëren.

2. **Trainingsplanner**:

- De trainingsplanner-interface wordt door de AI aangepast om specifiek aan te sluiten bij het ervaringsniveau en de beschikbare apparatuur van de gebruiker.
- Voor een beginner zonder apparatuur worden mogelijk eenvoudige lichaamsoefeningen getoond met gedetailleerde instructies en video's.
- Voor een gevorderde gebruiker met toegang tot een sportschool kunnen complexere routines worden weergegeven met minder uitleg.
- De inhoud van de trainingsplanner wordt niet simpelweg gefilterd uit een grote verzameling. Deze kan ter plekke worden gegenereerd op basis van een kennisbank die wordt bevraagd met context die alle bekende informatie over de gebruiker bevat.

3. **Voortgangsbewaking**:

- De voortgangsbewaking-interface evolueert op basis van de doelen en betrokkenheidspatronen van de gebruiker.

- Als een gebruiker voornamelijk gericht is op gewichtsverlies, toont de interface mogelijk prominent een gewichtstrend-grafiek en calorieverbranding-statistieken.
- Voor een gebruiker die spieren opbouwt, kunnen krachttoename en lichaamscompositie-veranderingen worden benadrukt.
- De AI kan dit deel van de applicatie aanpassen aan de daadwerkelijke voortgang van de gebruiker. Als de voortgang een tijd stilstaat, kan de app overschakelen naar een modus waarin wordt geprobeerd de gebruiker te verleiden om de redenen voor de tegenslag te onthullen, om deze vervolgens aan te pakken.

4. **Voedingsadvies**:

- Het voedingsgedeelte past zich aan aan de dieetvoorkeuren en -beperkingen van de gebruiker.
- Voor een veganistische gebruiker worden mogelijk plantaardige maaltijdsuggesties en eiwitbronnen getoond.
- Voor een gebruiker met glutenintolerantie worden automatisch glutenbevattende voedingsmiddelen uit de aanbevelingen gefilterd.
- Ook hier wordt de inhoud niet geput uit een enorme verzameling maaltijdgegevens die voor alle gebruikers geldt, maar wordt deze samengesteld uit een kennisbank die informatie bevat die aanpasbaar is op basis van de specifieke situatie en beperkingen van de gebruiker.
- Zo worden recepten gegenereerd met ingrediëntspecificaties die aansluiten bij de voortdurend veranderende caloriebehoeften van de gebruiker naarmate hun fitnessniveau en lichaamsstatistieken zich ontwikkelen.

5. **Motiverende Elementen**:

- De motiverende inhoud en meldingen van de app worden gepersonaliseerd op basis van het persoonlijkheidstype van de gebruiker en de reactie op verschillende motivatiestrategieën.

- Sommige gebruikers ontvangen bemoedigende berichten, terwijl anderen meer datagestuurde feedback krijgen.

In dit voorbeeld stelt GenUI FitAI in staat om een zeer gepersonaliseerde ervaring te creëren voor elke gebruiker, wat mogelijk de betrokkenheid, tevredenheid en de kans op het bereiken van fitnessdoelen vergroot. De interface-elementen, inhoud en zelfs de "persoonlijkheid" van de app passen zich aan om optimaal te voldoen aan de behoeften en voorkeuren van elke individuele gebruiker.

De Verschuiving naar Resultaatgericht Ontwerp

GenUI vertegenwoordigt een fundamentele verschuiving in de benadering van gebruikersinterface-ontwerp, van een focus op het creëren van specifieke interface-elementen naar een meer holistische, resultaatgerichte aanpak. Deze verschuiving heeft verschillende belangrijke implicaties:

1. **Focus op Gebruikersdoelen**:
    - Ontwerpers zullen dieper moeten nadenken over gebruikersdoelen en gewenste resultaten in plaats van specifieke interface-componenten.
    - De nadruk zal liggen op het creëren van systemen die interfaces kunnen genereren die gebruikers efficiënt en effectief helpen hun doelen te bereiken.
    - Er zullen nieuwe UI-frameworks ontstaan die AI-gebaseerde ontwerpers de tools geven die ze nodig hebben om gebruikerservaringen ter plekke en vanaf nul te kunnen genereren in plaats van op basis van vooraf gedefinieerde schermspecificaties.
2. **Veranderende Rol van Ontwerpers**:
    - Ontwerpers zullen overgaan van het maken van vaste layouts naar het definiëren van regels, beperkingen en richtlijnen die AI-systemen moeten volgen bij het genereren van interfaces.

- Ze zullen vaardigheden moeten ontwikkelen op gebieden zoals data-analyse, AI prompt engineering en systeemdenken om GenUI-systemen effectief te kunnen aansturen.

3. **Belang van Gebruikersonderzoek**:

   - Gebruikersonderzoek wordt nog belangrijker in een GenUI-context, omdat ontwerpers niet alleen gebruikersvoorkeuren moeten begrijpen, maar ook hoe deze voorkeuren en behoeften veranderen in verschillende contexten.
   - Continue gebruikerstests en feedbackloops zullen essentieel zijn om het vermogen van de AI om effectieve interfaces te genereren te verfijnen en te verbeteren.

4. **Ontwerpen voor Variabiliteit**:

   - In plaats van één "perfecte" interface te creëren, moeten ontwerpers rekening houden met meerdere mogelijke variaties en ervoor zorgen dat het systeem geschikte interfaces kan genereren voor diverse gebruikersbehoeften.
   - Dit omvat het ontwerpen voor randgevallen en het waarborgen dat de gegenereerde interfaces bruikbaarheid en toegankelijkheid behouden in verschillende configuraties.
   - Productdifferentiatie krijgt nieuwe dimensies door uiteenlopende perspectieven op gebruikerspsychologie en het benutten van unieke datasets en kennisbanken die niet beschikbaar zijn voor concurrenten.

Uitdagingen en Overwegingen

Hoewel GenUI spannende mogelijkheden biedt, brengt het ook verschillende uitdagingen en overwegingen met zich mee:

1. **Technische Beperkingen**:

- Huidige AI-technologie heeft, hoewel geavanceerd, nog steeds beperkingen in het begrijpen van complexe gebruikersintenties en het genereren van werkelijk contextbewuste interfaces.
- Prestatieproblemen gerelateerd aan real-time generatie van interface-elementen, vooral op minder krachtige apparaten.

2. **Gegevensvereisten**:

   - Afhankelijk van het gebruiksgeval kunnen effectieve GenUI-systemen aanzienlijke hoeveelheden gebruikersgegevens nodig hebben om gepersonaliseerde interfaces te genereren.
   - De uitdagingen bij het ethisch verzamelen van authentieke gebruikersgegevens roepen zorgen op over gegevensprivacy en beveiliging, evenals mogelijke vooroordelen in de data die gebruikt wordt om GenUI-modellen te trainen.

3. **Bruikbaarheid en Consistentie**:

   - Ten minste totdat de praktijk wijdverspreid wordt, kan een applicatie met constant veranderende interfaces leiden tot bruikbaarheidsproblemen, omdat gebruikers mogelijk moeite hebben met het vinden van vertrouwde elementen of efficiënt navigeren.
   - Het vinden van een balans tussen personalisatie en het behouden van een consistente, leerbare interface zal cruciaal zijn.

4. **Overmatige Afhankelijkheid van AI**:

   - Er bestaat een risico op het overdragen van te veel ontwerpbeslissingen aan AI-systemen, wat mogelijk kan leiden tot inspiratieloze, problematische of simpelweg niet-werkende interface-keuzes.
   - Menselijk toezicht en de mogelijkheid om AI-gegenereerde ontwerpen te overschrijven zullen in de nabije toekomst belangrijk blijven.

5. **Toegankelijkheidszorgen**:

    - Het waarborgen dat dynamisch gegenereerde interfaces toegankelijk blijven voor gebruikers met beperkingen brengt geheel nieuwe uitdagingen met zich mee, wat zorgwekkend is gezien het slechte niveau van toegankelijkheidsnaleving in typische systemen.
    - Aan de andere kant kunnen AI-ontwerpers worden geïmplementeerd met *ingebouwde* aandacht voor toegankelijkheid, en mogelijkheden om toegankelijke interfaces ter plekke te bouwen, net zoals ze UI bouwen voor gebruikers zonder beperkingen.
    - Hoe dan ook moeten GenUI-systemen worden ontworpen met robuuste toegankelijkheidsrichtlijnen en testprocessen.

6. **Gebruikersvertrouwen en Transparantie**:

    - Gebruikers kunnen zich ongemakkelijk voelen bij interfaces die “te veel lijken te weten” over hen of veranderen op manieren die ze niet begrijpen.
    - Het bieden van transparantie over hoe en waarom interfaces worden gepersonaliseerd zal belangrijk zijn voor het opbouwen van gebruikersvertrouwen.

Toekomstperspectief en Kansen

De toekomst van Generatieve UI (GenUI) heeft een enorme belofte voor het revolutionair veranderen van de manier waarop we interacteren met digitale producten en diensten. Naarmate deze technologie zich blijft ontwikkelen, kunnen we een ingrijpende verschuiving verwachten in hoe gebruikersinterfaces worden ontworpen, geïmplementeerd en ervaren. Ik denk dat GenUI het fenomeen is dat onze software eindelijk zal stuwen naar het gebied dat nu als sciencefiction wordt beschouwd.

Een van de meest opwindende vooruitzichten van GenUI is het potentieel om toegankelijkheid te verbeteren op een schaal die verder gaat dan alleen ervoor zorgen dat mensen met ernstige beperkingen niet volledig worden uitgesloten van het gebruik van software. Door interfaces automatisch aan te passen aan individuele gebruikersbehoeften, zou GenUI digitale ervaringen inclusiever kunnen maken dan ooit tevoren. Stel je interfaces voor die zich naadloos aanpassen om grotere tekst te bieden voor jongere of visueel beperkte gebruikers, of vereenvoudigde layouts voor mensen met cognitieve beperkingen, allemaal zonder handmatige configuratie of aparte "toegankelijke" versies van applicaties.

De personalisatiemogelijkheden van GenUI zullen waarschijnlijk leiden tot verhoogde gebruikersbetrokkenheid, tevredenheid en loyaliteit binnen een breed scala aan digitale producten. Naarmate interfaces beter worden afgestemd op individuele voorkeuren en gedragingen, zullen gebruikers digitale ervaringen intuïtiever en plezieriger vinden, wat mogelijk leidt tot diepere en betekenisvollere interacties met technologie.

GenUI heeft ook het potentieel om het onboardingproces voor nieuwe gebruikers te transformeren. Door intuïtieve, gepersonaliseerde eerste gebruikerservaringen te creëren die zich snel aanpassen aan het expertiseniveau van elke gebruiker, zou GenUI de leercurve van nieuwe applicaties aanzienlijk kunnen verminderen. Dit zou kunnen leiden tot snellere adoptie en meer gebruikersvertrouwen bij het verkennen van nieuwe functies en functionaliteiten.

Een andere opwindende mogelijkheid is het vermogen van GenUI om een consistente gebruikerservaring te behouden over verschillende apparaten en platforms, terwijl er wordt geoptimaliseerd voor elke specifieke gebruikscontext. Dit zou de langdurige uitdaging kunnen oplossen van het bieden van coherente ervaringen in een steeds meer gefragmenteerd apparatenlandschap, van smartphones en tablets tot desktopcomputers en opkomende technologieën zoals augmented reality-brillen.

De datagedreven aard van GenUI biedt kansen voor snelle iteratie en verbetering in UI-ontwerp. Door real-time gegevens te verzamelen over hoe gebruikers omgaan

met gegenereerde interfaces, kunnen ontwerpers en ontwikkelaars ongekende inzichten krijgen in gebruikersgedrag en voorkeuren. Deze feedbackloop zou kunnen leiden tot continue verbeteringen in UI-ontwerp, gedreven door werkelijke gebruikspatronen in plaats van aannames of beperkt gebruikersonderzoek.

Om zich voor te bereiden op deze verschuiving zullen ontwerpers hun vaardigheden en denkwijzen moeten ontwikkelen. De focus zal verschuiven van het creëren van vaste layouts naar het ontwikkelen van uitgebreide ontwerpsystemen en richtlijnen die AI-gestuurde interface-generatie kunnen informeren. Ontwerpers zullen een diep begrip moeten ontwikkelen van data-analyse, AI-technologieën en systeemdenken om GenUI-systemen effectief te kunnen begeleiden.

Bovendien zullen ontwerpers, naarmate GenUI de grenzen tussen ontwerp en technologie vervaagt, nauwer moeten samenwerken met ontwikkelaars en datawetenschappers. Deze interdisciplinaire aanpak zal cruciaal zijn bij het creëren van GenUI-systemen die niet alleen visueel aantrekkelijk en gebruiksvriendelijk zijn, maar ook technisch robuust en ethisch verantwoord.

De ethische implicaties van GenUI zullen ook op de voorgrond treden naarmate de technologie zich verder ontwikkelt. Ontwerpers zullen een cruciale rol spelen bij het ontwikkelen van raamwerken voor verantwoord AI-gebruik in interface-ontwerp, waarbij ze ervoor zorgen dat personalisatie de gebruikerservaringen verbetert zonder de privacy in gevaar te brengen of gebruikersgedrag op onethische wijze te manipuleren.

Als we naar de toekomst kijken, biedt GenUI zowel spannende mogelijkheden als belangrijke uitdagingen. Het heeft de potentie om meer intuïtieve, efficiënte en bevredigende digitale ervaringen te creëren voor gebruikers wereldwijd. Hoewel het van ontwerpers vraagt om zich aan te passen en nieuwe vaardigheden te verwerven, biedt het ook een ongekende kans om de toekomst van mens-computerinteractie op diepgaande en betekenisvolle manieren vorm te geven. De weg naar volledig gerealiseerde GenUI-systemen zal ongetwijfeld complex zijn, maar de potentiële voordelen op het gebied van verbeterde gebruikerservaringen en digitale

toegankelijkheid maken het een toekomst waar het waard is naar te streven.

Intelligente Werkstroomorganisatie

In het domein van applicatieontwikkeling spelen werkstromen een cruciale rol bij het definiëren van hoe taken, processen en gebruikersinteracties worden gestructureerd en uitgevoerd. Naarmate applicaties complexer worden en gebruikersverwachtingen blijven stijgen, wordt de behoefte aan intelligente en adaptieve werkstroomorganisatie steeds duidelijker.

De “Intelligente Werkstroomorganisatie” benadering richt zich op het benutten van AI-componenten om complexe werkstromen binnen applicaties dynamisch te organiseren en te optimaliseren. Het doel is om applicaties te creëren die efficiënter, responsiever en aanpasbaar zijn aan realtime gegevens en context.

In dit hoofdstuk zullen we de belangrijkste principes en patronen verkennen die ten grondslag liggen aan de intelligente werkstroomorganisatie-aanpak. We

zullen bekijken hoe AI kan worden gebruikt om taken intelligent te routeren, besluitvorming te automatiseren en werkstromen dynamisch aan te passen op basis van verschillende factoren zoals gebruikersgedrag, systeemprestaties en bedrijfsregels. Aan de hand van praktische voorbeelden en realistische scenario's zullen we het transformatieve potentieel van AI laten zien bij het stroomlijnen en optimaliseren van applicatiewerkstromen.

Of u nu bedrijfsapplicaties bouwt met ingewikkelde bedrijfsprocessen of consumentgerichte applicaties met dynamische gebruikerstrajecten, de patronen en technieken die in dit hoofdstuk worden besproken zullen u voorzien van de kennis en hulpmiddelen om intelligente en efficiënte werkstromen te creëren die de algehele gebruikerservaring verbeteren en bedrijfswaarde genereren.

Zakelijke Behoefte

Traditionele benaderingen van werkstroombeheer zijn vaak afhankelijk van vooraf gedefinieerde regels en statische beslisbomen, die rigide en inflexibel kunnen zijn en niet kunnen omgaan met het dynamische karakter van moderne applicaties.

Neem een scenario waarin een e-commerce applicatie een complex orderafhandelingsproces moet verwerken. De werkstroom kan meerdere stappen omvatten zoals ordervalidatie, voorraadcontrole, betalingsverwerking, verzending en klantmeldingen. Elke stap kan zijn eigen set regels, afhankelijkheden, externe integraties en uitzonderingsafhandelingsmechanismen hebben. Het handmatig beheren van een dergelijke werkstroom of via hardgecodeerde logica kan snel omslachtig, foutgevoelig en moeilijk te onderhouden worden.

Bovendien moet de werkstroom zich mogelijk aanpassen en zichzelf optimaliseren op basis van realtime gegevens en systeemprestaties naarmate de applicatie schaalt en het aantal gelijktijdige gebruikers groeit. Tijdens piekverkeersperiodes moet de applicatie bijvoorbeeld de werkstroom dynamisch aanpassen om bepaalde taken te prioriteren, bronnen efficiënt toe te wijzen en een soepele gebruikerservaring te garanderen.

Dit is waar de “Intelligente Werkstroomorganisatie” benadering in beeld komt. Door gebruik te maken van AI-componenten kunnen ontwikkelaars werkstromen creëren die intelligent, adaptief en zelf-optimaliserend zijn. AI kan grote hoeveelheden gegevens analyseren, leren van eerdere ervaringen en geïnformeerde beslissingen nemen in realtime om de werkstroom effectief te organiseren.

Belangrijkste Voordelen

1. **Verhoogde Efficiëntie**: AI kan taaktoewijzing, brongebruik en werkstroomuitvoering optimaliseren, wat leidt tot snellere verwerkingstijden en verbeterde algehele efficiëntie.
2. **Aanpasbaarheid**: Door AI aangestuurde werkstromen kunnen zich dynamisch aanpassen aan veranderende omstandigheden, zoals schommelingen in gebruikersvraag, systeemprestaties of bedrijfsvereisten, waardoor de applicatie responsief en veerkrachtig blijft.
3. **Geautomatiseerde Besluitvorming**: AI kan complexe besluitvormingsprocessen binnen de werkstroom automatiseren, waardoor handmatige interventie wordt verminderd en het risico op menselijke fouten wordt geminimaliseerd.
4. **Personalisatie**: AI kan gebruikersgedrag, voorkeuren en context analyseren om de werkstroom te personaliseren en op maat gemaakte ervaringen te leveren aan individuele gebruikers.
5. **Schaalbaarheid**: Door AI aangedreven werkstromen kunnen naadloos schalen om toenemende volumes van gegevens en gebruikersinteracties te verwerken, zonder concessies te doen aan prestaties of betrouwbaarheid.

In de volgende secties zullen we de belangrijkste patronen en technieken verkennen die de implementatie van intelligente werkstromen mogelijk maken en laten we praktijkvoorbeelden zien van hoe AI het werkstroombeheer in moderne applicaties transformeert.

Belangrijke Patronen

Om intelligente werkstroomorganisatie in applicaties te implementeren, kunnen ontwikkelaars gebruik maken van verschillende belangrijke patronen die de kracht van AI benutten. Deze patronen bieden een gestructureerde aanpak voor het ontwerpen en beheren van werkstromen, waardoor applicaties processen kunnen aanpassen, optimaliseren en automatiseren op basis van realtime gegevens en context. Laten we enkele van de fundamentele patronen in intelligente werkstroomorganisatie verkennen.

Dynamische Taakroutering

Dit patroon omvat het gebruik van AI om taken binnen een werkstroom intelligent te routeren op basis van verschillende factoren zoals taakprioriteit, beschikbaarheid van bronnen en systeemprestaties. AI-algoritmen kunnen de kenmerken van elke taak analyseren, rekening houden met de huidige staat van het systeem en geïnformeerde beslissingen nemen om taken toe te wijzen aan de meest geschikte bronnen of verwerkingspaden. Dynamische taakroutering zorgt ervoor dat taken efficiënt worden verdeeld en uitgevoerd, waardoor de algehele werkstroomprestaties worden geoptimaliseerd.

```
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, # ...
```

```
15    function :assess_system_performance, # ...
16    function :assign_task_to_resource, # ...
17
18    DIRECTIVE = "You are a task router, responsible for intelligently
19     assigning tasks to available resources based on priority, resource
20     availability, and system performance..."
21
22    def initialize(task)
23      self.task = task
24      transcript << { system: DIRECTIVE }
25      transcript << { user: task.to_json }
26    end
27
28    def perform
29      while task.unassigned?
30        chat_completion
31
32        # todo: add max loop counter and break
33      end
34
35      # capture the transcript for later analysis
36      task.update(routing_transcript: transcript)
37    end
38  end
```

Let op de lus die wordt gecreëerd door de `while` expressie op regel 29, die de AI blijft vragen totdat de taak is toegewezen. Op regel 35 slaan we het transcript van de taak op voor latere analyse en debugging, mocht dit nodig zijn.

Contextuele Besluitvorming

Je kunt vergelijkbare code gebruiken om contextbewuste beslissingen binnen een workflow te nemen. Door het analyseren van relevante gegevenspunten zoals gebruikersvoorkeuren, historische patronen, en realtime invoer, kunnen AI-componenten de meest geschikte handelwijze bepalen bij elk beslissingspunt in de workflow. Pas het gedrag van je workflow aan op basis van de specifieke context van elke gebruiker of scenario, voor gepersonaliseerde en geoptimaliseerde ervaringen.

Adaptieve Workflowcompositie

Dit patroon richt zich op het dynamisch samenstellen en aanpassen van workflows op basis van veranderende eisen of omstandigheden. AI kan de huidige staat van de workflow analyseren, knelpunten of inefficiënties identificeren, en automatisch de workflowstructuur aanpassen om de prestaties te optimaliseren. Adaptieve workflowcompositie stelt applicaties in staat om continu te evolueren en hun processen te verbeteren zonder handmatige interventie.

Uitzonderingsafhandeling en Herstel

Uitzonderingsafhandeling en herstel zijn kritieke aspecten van intelligente workfloworganisatie. Bij het werken met AI-componenten en complexe workflows is het essentieel om uitzonderingen te anticiperen en elegant af te handelen om de stabiliteit en betrouwbaarheid van het systeem te waarborgen.

Hier zijn enkele belangrijke overwegingen en technieken voor uitzonderingsafhandeling en herstel in intelligente workflows:

1. **Uitzonderingspropagatie**: Implementeer een consistente aanpak voor het propageren van uitzonderingen tussen workflowcomponenten. Wanneer een uitzondering zich voordoet binnen een component, moet deze worden opgevangen, gelogd en doorgegeven aan de orchestrator of een afzonderlijke component die verantwoordelijk is voor het afhandelen van uitzonderingen. Het idee is om uitzonderingsafhandeling te centraliseren en te voorkomen dat uitzonderingen stilzwijgend worden onderdrukt, en tevens mogelijkheden te openen voor Intelligente Foutafhandeling.
2. **Herhaalingsmechanismen**: Herhaalingsmechanismen helpen de veerkracht van de workflow te verbeteren en behandelen tijdelijke storingen op elegante wijze. Implementeer zeker herhaalingsmechanismen voor tijdelijke

of herstelbare uitzonderingen, zoals problemen met netwerkconnectiviteit of onbeschikbaarheid van bronnen die automatisch opnieuw kunnen worden geprobeerd na een bepaalde vertraging. Met een AI-gestuurde orchestrator of uitzonderingsafhandelaar hoeven je herhaalsstrategieën niet mechanisch van aard te zijn, vertrouwend op vaste algoritmen zoals exponentiële terugval. Je kunt de afhandeling van de herhaling overlaten aan het "oordeel" van de AI-component die verantwoordelijk is voor het beslissen hoe de uitzondering moet worden afgehandeld.

3. **Terugvalstrategieën**: Als een AI-component er niet in slaagt een geldig antwoord te geven of een fout tegenkomt—een veel voorkomende gebeurtenis gezien zijn geavanceerde aard—zorg dan voor een terugvalmechanisme om ervoor te zorgen dat de workflow kan doorgaan. Dit kan het gebruik van standaardwaarden, alternatieve algoritmen of een Mens in de Loop omvatten om beslissingen te nemen en de workflow vooruit te helpen.
4. **Compenserende Acties**: De richtlijnen van de orchestrator moeten instructies bevatten over compenserende acties om uitzonderingen af te handelen die niet automatisch kunnen worden opgelost. Compenserende acties zijn stappen die worden ondernomen om de effecten van een mislukte operatie ongedaan te maken of te verzachten. Als bijvoorbeeld een betalingsverwerkingsstap mislukt, zou een compenserende actie kunnen zijn om de transactie terug te draaien en de gebruiker te informeren. Compenserende acties helpen bij het behouden van dataconsistentie en integriteit bij uitzonderingen.
5. **Uitzonderingsmonitoring en Waarschuwingen**: Zet monitoring- en waarschuwingsmechanismen op om kritieke uitzonderingen te detecteren en relevante belanghebbenden hierover te informeren. De orchestrator kan bewust worden gemaakt van drempels en regels om waarschuwingen te activeren wanneer uitzonderingen bepaalde limieten overschrijden of wanneer specifieke soorten uitzonderingen optreden. Dit maakt proactieve identificatie en oplossing van problemen mogelijk voordat ze het hele systeem beïnvloeden.

Hier is een voorbeeld van uitzonderingsafhandeling en herstel in een Ruby workflow-component:

```
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
```

In dit voorbeeld controleert de `InventoryManager`-component de beschikbaarheid van een product voor een bepaalde bestelling. Als de beschikbare hoeveelheid onvoldoende is, wordt er een `InsufficientInventoryError` gegenereerd. De exceptie wordt opgevangen, gelogd en er wordt een hertryingsmechanisme geïmplementeerd. Als de hertryingslimiet wordt overschreden, schakelt de component over op handmatige interventie door een beheerder te waarschuwen.

Door robuuste exceptieafhandeling en herstelmechanismen te implementeren, kunt u ervoor zorgen dat uw intelligente workflows veerkrachtig en onderhoudbaar zijn, en onverwachte situaties elegant kunnen afhandelen.

Deze patronen vormen de basis van intelligente workflow-orchestratie en kunnen worden gecombineerd en aangepast aan de specifieke vereisten van verschillende applicaties. Door gebruik te maken van deze patronen kunnen ontwikkelaars workflows creëren die flexibel en veerkrachtig zijn, en geoptimaliseerd voor prestaties en gebruikerservaring.

In het volgende gedeelte zullen we onderzoeken hoe deze patronen in de praktijk kunnen worden geïmplementeerd, waarbij we gebruik maken van praktijkvoorbeelden en codefragmenten om de integratie van AI-componenten in workflowbeheer te illustreren.

Implementatie van Intelligente Workflow-orchestratie in de Praktijk

Nu we de belangrijkste patronen in intelligente workflow-orchestratie hebben verkend, laten we eens kijken hoe deze patronen kunnen worden geïmplementeerd in praktijktoepassingen. We zullen praktische voorbeelden en codefragmenten geven om de integratie van AI-componenten in workflowbeheer te illustreren.

Intelligente Orderverwerker

Laten we een praktisch voorbeeld bekijken van het implementeren van intelligente workflow-orchestratie met behulp van een AI-gestuurde `OrderProcessor`-component in een Ruby on Rails e-commerce applicatie. De `OrderProcessor` implementeert het Process Manager Enterprise Integration concept dat we voor het eerst tegenkwamen

in Hoofdstuk 3 bij de bespreking van Multitude of Workers. De component is verantwoordelijk voor het beheren van de orderverwerkingsworkflow, het nemen van routeringsbeslissingen op basis van tussenresultaten, en het orchestreren van de uitvoering van verschillende verwerkingsstappen.

Het orderverwerkingsproces bestaat uit meerdere stappen zoals ordervalidatie, voorraadcontrole, betalingsverwerking en verzending. Elke stap wordt geïmplementeerd als een apart werkproces dat een specifieke taak uitvoert en het resultaat teruggeeft aan de `OrderProcessor`. De stappen zijn niet verplicht en hoeven zelfs niet noodzakelijkerwijs in een bepaalde volgorde te worden uitgevoerd.

Hier is een voorbeeldimplementatie van de `OrderProcessor`. Het bevat twee mixins van Raix. De eerste (`ChatCompletion`) geeft het de mogelijkheid om chatberichten te voltooien, wat dit een AI-component maakt. De tweede (`FunctionDispatch`) maakt functie-aanroep door de AI mogelijk, waardoor deze op een prompt kan reageren met een functie-aanroep in plaats van een tekstbericht.

De werkerfuncties (`validate_order`, `check_inventory`, et al) delegeren naar hun respectievelijke werkerklassen, die AI- of niet-AI-componenten kunnen zijn, met als enige vereiste dat ze de resultaten van hun werk teruggeven in een formaat dat als tekst kan worden weergegeven.

Net als bij alle andere voorbeelden in dit deel van het boek is deze code in feite pseudo-code en is alleen bedoeld om de betekenis van het patroon over te brengen en uw eigen creaties te inspireren. Volledige beschrijvingen van patronen en complete codevoorbeelden zijn opgenomen in Deel 2.

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

  SYSTEM_DIRECTIVE = "You are an order processor, tasked with..."

  def initialize(order)
    self.order = order
    transcript << { system: SYSTEM_DIRECTIVE }
    transcript << { user: order.to_json }
  end

  def perform
    # will continue looping until `stop_looping!` is called
    chat_completion(loop: true)
  end

  # list of functions available to be called by the AI
  # truncated for brevity

  def functions
    [
      {
        name: "validate_order",
        description: "Invoke to check validity of order",
        parameters: {
          ...
      },
      ...
    ]
  end

  # implementation of functions that can be called by the AI
  # entirely at its discretion, depending on the needs of the order

  def validate_order
    OrderValidationWorker.perform(@order)
  end

  def check_inventory
    InventoryCheckWorker.perform(@order)
  end
```

```
43
44    def process_payment
45      PaymentProcessingWorker.perform(@order)
46    end
47
48    def schedule_shipping
49      ShippingSchedulerWorker.perform(@order)
50    end
51
52    def send_confirmation
53      OrderConfirmationWorker.perform(@order)
54    end
55
56    def finished_processing
57      @order.update!(transcript:, processed_at: Time.current)
58      stop_looping!
59    end
60  end
```

In het voorbeeld wordt de OrderProcessor geïnitialiseerd met een orderobject en houdt deze een transcript bij van de werkstroomuitvoering, in het typische conversatietranscriptformaat dat eigen is aan grote taalmodellen. De AI krijgt volledige controle over het orchestreren van de uitvoering van verschillende verwerkingsstappen, zoals ordervalidatie, voorraadcontrole, betalingsverwerking en verzending.

Elke keer dat de `chat_completion` methode wordt aangeroepen, wordt het transcript naar de AI gestuurd om een voltooiing als functieaanroep te leveren. Het is volledig aan de AI om het resultaat van de vorige stap te analyseren en de juiste actie te bepalen. Als bijvoorbeeld de voorraadcontrole lage voorraadniveaus laat zien, kan de `OrderProcessor` een aanvullingstaak plannen. Als de betalingsverwerking mislukt, kan deze een nieuwe poging initiëren of de klantenservice op de hoogte stellen.

> Het bovenstaande voorbeeld heeft geen functies gedefinieerd voor aanvulling of het informeren van de klantenservice, maar dat zou absoluut kunnen.

Het transcript groeit elke keer dat een functie wordt aangeroepen en dient als een registratie van de werkstroomuitvoering, inclusief de resultaten van elke stap en de door AI gegenereerde instructies voor de volgende stappen. Dit transcript kan worden gebruikt voor debugging, auditing en het verschaffen van inzicht in het orderafhandelingsproces.

Door AI te benutten in de `OrderProcessor`, kan de e-commerce applicatie de werkstroom dynamisch aanpassen op basis van realtime gegevens en uitzonderingen intelligent afhandelen. De AI-component kan geïnformeerde beslissingen nemen, de werkstroom optimaliseren en zorgen voor een soepele orderverwerking, zelfs in complexe scenario's.

Het feit dat de enige vereiste voor de werkprocessen is om een begrijpelijke output te retourneren voor de AI om te overwegen bij het beslissen wat de volgende stap moet zijn, zou je kunnen doen beseffen hoe deze aanpak het werk van input/output-mapping kan verminderen dat typisch nodig is bij het integreren van verschillende systemen met elkaar.

Intelligente Content Moderator

Sociale media-applicaties vereisen over het algemeen ten minste minimale contentmoderatie om een veilige en gezonde community te waarborgen. Dit voorbeeld van een `ContentModerator` component maakt gebruik van AI om de moderatiewerkstroom intelligent te orchestreren, waarbij beslissingen worden genomen op basis van de kenmerken van de content en de resultaten van verschillende moderatiestappen.

Het moderatieproces omvat meerdere stappen zoals tekstanalyse, beeldherkenning, beoordeling van gebruikersreputatie en handmatige beoordeling. Elke stap wordt geïmplementeerd als een afzonderlijk werkproces dat een specifieke taak uitvoert en het resultaat terugstuurt naar de `ContentModerator`.

Hier is een voorbeeldimplementatie van de `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
```

In dit voorbeeld wordt de `ContentModerator` geïnitialiseerd met een content-object en houdt het een moderatie-transcript bij in het gesprekformaat. Het AI-component heeft volledige controle over de moderatie-werkstroom en bepaalt welke stappen er uitgevoerd moeten worden op basis van de kenmerken van de content en de resultaten van elke stap.

De beschikbare werkerfuncties die de AI kan aanroepen zijn onder andere `analyze_text`, `recognize_image`, `assess_user_reputation`, en `escalate_to_manual_review`. Elke functie delegeert de taak naar een corresponderende werkerproces (`TextAnalysisWorker`, `ImageRecognitionWorker`, etc.) en voegt het resultaat toe aan het moderatie-transcript, met uitzondering van de escalatiefunctie, die fungeert als eindstatus. Tot slot fungeren de functies `approve_content` en `reject_content` ook als eindstatussen.

Het AI-component analyseert de content en bepaalt welke actie geschikt is. Als de content afbeeldingsreferenties bevat, kan het de `recognize_image`-werker aanroepen voor hulp bij een visuele beoordeling. Als een werker waarschuwt

voor mogelijk schadelijke content, kan de AI beslissen om de content te escaleren voor handmatige beoordeling of deze direct af te wijzen. Maar afhankelijk van de ernst van de waarschuwing, kan de AI ervoor kiezen om de resultaten van de gebruikersreputatiebeoordeling te gebruiken bij het beslissen hoe om te gaan met content waarover het anders niet zeker is. Afhankelijk van het gebruiksgeval hebben vertrouwde gebruikers mogelijk meer speelruimte in wat ze kunnen plaatsen. Enzovoort, enzovoort…

Net als bij het vorige voorbeeld van de procesmanager dient het moderatie-transcript als een registratie van de werkstroomuitvoering, inclusief de resultaten van elke stap en de door AI gegenereerde beslissingen. Dit transcript kan worden gebruikt voor auditing, transparantie en het verbeteren van het moderatieproces in de loop der tijd.

Door AI te benutten in de `ContentModerator` kan de social media-applicatie de moderatie-werkstroom dynamisch aanpassen op basis van de kenmerken van de content en complexe moderatiescenario's intelligent afhandelen. Het AI-component kan weloverwogen beslissingen nemen, de werkstroom optimaliseren en zorgen voor een veilige en gezonde community-ervaring.

Laten we twee andere voorbeelden bekijken die voorspellende taakplanning en foutafhandeling en -herstel demonstreren binnen de context van intelligente werkstroomorganisatie.

Voorspellende Taakplanning in een Klantenondersteuningssysteem

In een klantenondersteuningsapplicatie gebouwd met Ruby on Rails is het efficiënt beheren en prioriteren van supporttickets cruciaal voor het bieden van tijdige hulp aan klanten. Het `SupportTicketScheduler`-component maakt gebruik van AI om voorspellend supporttickets in te plannen en toe te wijzen aan beschikbare medewerkers op basis van verschillende factoren zoals ticketurgentie, expertise van de medewerker en werklast.

```
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",
        # ...
```

```ruby
      },
      {
        name: "reschedule_ticket",
        # ...
      }
    ]
  end

  # implementation of functions that can be called by the AI
  # entirely at its discretion, depending on the needs of the order

  def analyze_ticket_urgency
    result = TicketUrgencyAnalyzer.perform(@ticket)
    continue_with(result)
  end

  def list_available_agents
    result = ListAvailableAgents.perform
    continue_with(result)
  end

  def predict_agent_workload
    result = AgentWorkloadPredictor.perform
    continue_with(result)
  end

  def assign_ticket_to_agent
    TicketAssigner.perform(@ticket, @transcript)
  end

  def delay_assignment(until)
    until = DateTimeStandardizer.process(until)
    SupportTicketScheduler.delay(@ticket, @transcript, until)
  end

  private

  def continue_with(result)
    @transcript << { function: result }
    complete(@transcript)
  end
end
```

In dit voorbeeld wordt de `SupportTicketScheduler` geïnitialiseerd met een supportticket-object en houdt deze een planningsverslag bij. Het AI-component analyseert de ticketdetails en plant voorspellend de tickettoewijzing op basis van factoren zoals ticketurgentie, agentexpertise en voorspelde werkbelasting van de agent.

De beschikbare functies die de AI kan aanroepen zijn `analyze_ticket_urgency`, `list_available_agents`, `predict_agent_workload` en `assign_ticket_to_agent`. Elke functie delegeert de taak naar een corresponderende analyzer- of predictor-component en voegt het resultaat toe aan het planningsverslag. De AI heeft ook de mogelijkheid om toewijzing uit te stellen met behulp van de `delay_assignment` functie.

Het AI-component onderzoekt het planningsverslag en neemt weloverwogen beslissingen over tickettoewijzing. Het houdt rekening met de urgentie van het ticket, de expertise van beschikbare agents en de voorspelde werkbelasting van elke agent om de meest geschikte agent voor de afhandeling van het ticket te bepalen.

Door gebruik te maken van voorspellende taakplanning kan de klantenondersteuningsapplicatie de tickettoewijzing optimaliseren, reactietijden verkorten en de algehele klanttevredenheid verbeteren. Proactief en efficiënt beheer van supporttickets zorgt ervoor dat de juiste tickets op het juiste moment aan de juiste agents worden toegewezen.

Foutafhandeling en Herstel in een Gegevensverwerkingspijplijn

Het afhandelen van uitzonderingen en herstellen van fouten is essentieel om gegevensintegriteit te waarborgen en gegevensverlies te voorkomen. De `DataProcessingOrchestrator`-component maakt gebruik van AI om op intelligente wijze uitzonderingen af te handelen en het herstelproces te orchestreren in een gegevensverwerkingspijplijn

```
class DataProcessingOrchestrator
  include Raix::ChatCompletion
  include Raix::FunctionDispatch

  SYSTEM_DIRECTIVE = "You are a data processing orchestrator..."

  def initialize(data_batch)
    @data_batch = data_batch
    @transcript = [
      { system: SYSTEM_DIRECTIVE },
      { user: data_batch.to_json }
    ]
  end

  def perform
    complete(@transcript)
  end

  def model
    "openai/gpt-4"
  end

  def functions
    [
      {
        name: "validate_data",
        # ...
      },
      {
        name: "process_data",
        # ...
      },
      {
        name: "request_fix",
        # ...
      },
      {
        name: "retry_processing",
        # ...
      },
      {
        name: "mark_data_as_failed",
```

```
        # ...
      },
      {
        name: "finished",
        # ...
      }
    ]
  end

  # implementation of functions that can be called by the AI
  # entirely at its discretion, depending on the needs of the order

  def validate_data
    result = DataValidator.perform(@data_batch)
    continue_with(result)
  rescue ValidationException => e
    handle_validation_exception(e)
  end

  def process_data
    result = DataProcessor.perform(@data_batch)
    continue_with(result)
  rescue ProcessingException => e
    handle_processing_exception(e)
  end

  def request_fix(description_of_fix)
    result = SmartDataFixer.new(description_of_fix, @data_batch)
    continue_with(result)
  end

  def retry_processing(timeout_in_seconds)
    wait(timeout_in_seconds)
    process_data
  end

  def mark_data_as_failed
    @data_batch.update!(status: 'failed', transcript: @transcript)
  end

  def finished
    @data_batch.update!(status: 'finished', transcript: @transcript)
```

```
  end

  private

  def continue_with(result)
    @transcript << { function: result }
    complete(@transcript)
  end

  def handle_validation_exception(exception)
    @transcript << { exception: exception.message }
    complete(@transcript)
  end

  def handle_processing_exception(exception)
    @transcript << { exception: exception.message }
    complete(@transcript)
  end
end
```

In dit voorbeeld wordt de `DataProcessingOrchestrator` geïnitialiseerd met een databatch-object en houdt deze een verwerkingstranscript bij. Het AI-component orkestreert de dataverwerkingspijplijn, handelt excepties af en herstelt van fouten waar nodig.

De beschikbare functies die de AI kan aanroepen zijn `validate_data`, `process_data`, `request_fix`, `retry_processing`, en `mark_data_as_failed`. Elke functie delegeert de taak naar een corresponderend dataverwerkingscomponent en voegt het resultaat of de exceptiedetails toe aan het verwerkingstranscript.

Als er een validatie-exceptie optreedt tijdens de `validate_data` stap, voegt de `handle_validation_exception` functie de exceptiegegevens toe aan het transcript en geeft de controle terug aan de AI. Ook als er een verwerkingsexceptie optreedt tijdens de `process_data` stap, kan de AI beslissen over de herstelstrategie.

Afhankelijk van de aard van de opgetreden exceptie kan de AI naar eigen inzicht beslissen om `request_fix` aan te roepen, wat delegeert naar een AI-aangedreven

`SmartDataFixer` component (zie hoofdstuk over Zelf-herstellende Data). De data fixer krijgt een gewone Nederlandse beschrijving van hoe het de `@data_batch` moet aanpassen zodat de verwerking opnieuw kan worden geprobeerd. Misschien zou een succesvolle nieuwe poging inhouden dat records die de validatie niet hebben doorstaan uit de databatch worden verwijderd en/of worden gekopieerd naar een andere verwerkingspijplijn voor menselijke controle? De mogelijkheden zijn bijna eindeloos.

Door AI-gestuurde exceptie-afhandeling en herstel te integreren, wordt de dataverwerkingsapplicatie veerkrachtiger en fouttoleranter. De `DataProcessingOrchestrator` beheert op intelligente wijze excepties, minimaliseert gegevensverlies en zorgt voor een soepele uitvoering van de dataverwerkingsworkflow.

Monitoring en Logging

Monitoring en logging bieden inzicht in de voortgang, prestaties en gezondheid van AI-aangedreven workflowcomponenten, waardoor ontwikkelaars het gedrag van het systeem kunnen volgen en analyseren. Het implementeren van effectieve monitoring- en loggingmechanismen is essentieel voor het debuggen, auditen en continue verbeteren van intelligente workflows.

Monitoren van Workflowvoortgang en Prestaties

Om de soepele uitvoering van intelligente workflows te waarborgen, is het belangrijk om de voortgang en prestaties van elk workflowcomponent te monitoren. Dit omvat het bijhouden van belangrijke metrics en gebeurtenissen gedurende de workflow-levenscyclus.

Enkele belangrijke aspecten om te monitoren zijn:

1. **Workflow Uitvoeringstijd**: Meet de tijd die elk workflowcomponent nodig heeft

om zijn taak te voltooien. Dit helpt bij het identificeren van prestatie-knelpunten en het optimaliseren van de algehele workflow-efficiëntie.

2. **Brongebruik**: Monitor het gebruik van systeembronnen, zoals CPU, geheugen en opslag, door elk workflowcomponent. Dit helpt ervoor te zorgen dat het systeem binnen zijn capaciteit werkt en de werklast effectief kan verwerken.

3. **Foutpercentages en Excepties**: Volg het voorkomen van fouten en excepties binnen workflowcomponenten. Dit helpt bij het identificeren van potentiële problemen en maakt proactieve foutafhandeling en herstel mogelijk.

4. **Beslispunten en Uitkomsten**: Monitor de beslispunten binnen de workflow en de uitkomsten van AI-gestuurde beslissingen. Dit geeft inzicht in het gedrag en de effectiviteit van de AI-componenten.

De gegevens die door monitoringprocessen worden verzameld, kunnen worden weergegeven in dashboards of worden gebruikt als input voor geplande rapporten die systeembeheerders informeren over de gezondheid van het systeem.

Monitoringgegevens kunnen worden doorgegeven aan een AI-aangedreven systeembeheerderproces voor beoordeling en mogelijke actie!

Loggen van Belangrijke Gebeurtenissen en Beslissingen

Loggen is een essentiële praktijk die bestaat uit het vastleggen en opslaan van relevante informatie over belangrijke gebeurtenissen, beslissingen en excepties die optreden tijdens de workflow-uitvoering.

Enkele belangrijke aspecten om te loggen zijn:

1. **Workflow Initiatie en Voltooiing**: Log de start- en eindtijden van elke workflow-instantie, samen met relevante metadata zoals de invoergegevens en gebruikerscontext.

2. **Component Uitvoering**: Log de uitvoeringsdetails van elk workflowcomponent, inclusief de invoerparameters, uitvoerresultaten en eventuele gegenereerde tussenliggende gegevens.

3. **AI-beslissingen en Redenering**: Log de beslissingen die door AI-componenten worden genomen, samen met de onderliggende redenering of betrouwbaarheidsscores. Dit zorgt voor transparantie en maakt het mogelijk AI-gestuurde beslissingen te auditen.

4. **Excepties en Foutmeldingen**: Log eventuele excepties of foutmeldingen die tijdens de workflow-uitvoering worden aangetroffen, inclusief de stack trace en relevante contextinformatie.

Loggen kan worden geïmplementeerd met verschillende technieken, zoals schrijven naar logbestanden, logs opslaan in een database of logs verzenden naar een gecentraliseerde loggingservice. Het is belangrijk om een logging-framework te kiezen dat flexibiliteit, schaalbaarheid en eenvoudige integratie met de applicatiearchitectuur biedt.

Hier is een voorbeeld van hoe logging kan worden geïmplementeerd in een Ruby on Rails-applicatie met behulp van de `ActiveSupport::Logger` klasse:

```ruby
class WorkflowLogger
  def self.log(message, severity = :info)
    @logger ||= ActiveSupport::Logger.new('workflow.log')
    @logger.formatter ||= proc do |severity, datetime, progname, msg|
      "#{datetime} [#{severity}] #{msg}\n"
    end
    @logger.send(severity, message)
  end
end

# Usage example
WorkflowLogger.log("Workflow initiated for order ##{@order.id}")
WorkflowLogger.log("Payment processing completed successfully")
WorkflowLogger.log("Inventory check failed for item ##{item.id}", :error)
```

Door strategisch logboekregistraties te plaatsen in de workflow-componenten en AI-beslispunten, kunnen ontwikkelaars waardevolle informatie vastleggen voor debugging,

auditing en analyse.

Voordelen van Monitoring en Logging

Het implementeren van monitoring en logging in intelligente workfloworchestratie biedt verschillende voordelen:

1. **Debuggen en Probleemoplossing**: Gedetailleerde logboeken en monitoringgegevens helpen ontwikkelaars bij het snel identificeren en diagnosticeren van problemen. Ze bieden inzicht in de uitvoering van de workflow, componentinteracties en eventuele fouten of uitzonderingen die zich voordoen.

2. **Prestatie-optimalisatie**: Het monitoren van prestatiemetrieken stelt ontwikkelaars in staat knelpunten te identificeren en workflowcomponenten te optimaliseren voor betere efficiëntie. Door het analyseren van uitvoeringstijden, resourcegebruik en andere metrieken kunnen ontwikkelaars weloverwogen beslissingen nemen om de algehele prestaties van het systeem te verbeteren.

3. **Auditing en Compliance**: Het registreren van belangrijke gebeurtenissen en beslissingen zorgt voor een auditspoor voor regelgeving en verantwoording. Het stelt organisaties in staat om de acties van AI-componenten te volgen en te verifiëren en zorgt voor naleving van bedrijfsregels en wettelijke vereisten.

4. **Continue Verbetering**: Monitoring- en logginggegevens dienen als waardevolle input voor continue verbetering van intelligente workflows. Door het analyseren van historische gegevens, het identificeren van patronen en het meten van de effectiviteit van AI-beslissingen kunnen ontwikkelaars de workfloworchestratie-logica iteratief verfijnen en verbeteren.

Overwegingen en Best Practices

Bij het implementeren van monitoring en logging in intelligente workfloworchestratie moet rekening worden gehouden met de volgende best practices:

1. **Definieer Duidelijke Monitoringmetrieken**: Identificeer de belangrijkste metrieken en gebeurtenissen die gemonitord moeten worden op basis van de specifieke vereisten van de workflow. Focus op metrieken die betekenisvolle inzichten geven in de prestaties, gezondheid en het gedrag van het systeem.

2. **Implementeer Gedetailleerde Logging**: Zorg ervoor dat logboekregistraties op de juiste punten binnen de workflowcomponenten en AI-beslispunten worden geplaatst. Leg relevante contextinformatie vast, zoals invoerparameters, uitvoerresultaten en eventuele gegenereerde tussenliggende gegevens.

3. **Gebruik Gestructureerde Logging**: Adopteer een gestructureerd loggingformaat om het parseren en analyseren van loggegevens te vergemakkelijken. Gestructureerde logging maakt betere zoekbaarheid, filtering en aggregatie van logboekitems mogelijk.

4. **Beheer Logbehoud en -rotatie**: Implementeer beleid voor logbehoud en -rotatie om de opslag en levenscyclus van logbestanden te beheren. Bepaal de juiste bewaartermijn op basis van wettelijke vereisten, opslagbeperkingen en analysebehoeften. Indien mogelijk, besteed logging uit aan een externe dienst zoals Papertrail.

5. **Beveilig Gevoelige Informatie**: Wees voorzichtig bij het loggen van gevoelige informatie, zoals persoonlijk identificeerbare informatie (PII) of vertrouwelijke bedrijfsgegevens. Implementeer passende beveiligingsmaatregelen, zoals gegevensmasking of encryptie, om gevoelige informatie in logbestanden te beschermen.

6. **Integreer met Monitoring- en Waarschuwingstools**: Maak gebruik van monitoring- en waarschuwingstools om het verzamelen, analyseren en visualiseren van monitoring- en logginggegevens te centraliseren. Deze tools kunnen realtime inzichten bieden, waarschuwingen genereren op basis van vooraf gedefinieerde drempels en proactieve probleemdetectie en -oplossing faciliteren. Mijn favoriete tool hiervoor is Datadog.

Door uitgebreide monitoring- en loggingmechanismen te implementeren, kunnen ontwikkelaars waardevolle inzichten krijgen in het gedrag en de prestaties van

intelligente workflows. Deze inzichten maken effectief debuggen, optimalisatie en continue verbetering van AI-gestuurde workfloworchestratie-systemen mogelijk.

Schaalbaarheid en Prestatieoverwegingen

Schaalbaarheid en prestaties zijn kritieke aspecten om rekening mee te houden bij het ontwerpen en implementeren van intelligente workfloworchestratie-systemen. Naarmate het volume van gelijktijdige workflows en de complexiteit van AI-gestuurde componenten toenemen, wordt het essentieel om ervoor te zorgen dat het systeem de werklast efficiënt kan verwerken en naadloos kan schalen om aan groeiende eisen te voldoen.

Omgaan met Grote Volumes Gelijktijdige Workflows

Intelligente workfloworchestratie-systemen moeten vaak een groot aantal gelijktijdige workflows verwerken. Overweeg de volgende strategieën om schaalbaarheid te waarborgen:

1. **Asynchrone Verwerking**: Implementeer asynchrone verwerkingsmechanismen om de uitvoering van workflowcomponenten te ontkoppelen. Dit stelt het systeem in staat om meerdere workflows gelijktijdig te verwerken zonder te blokkeren of te wachten tot elke component is voltooid. Asynchrone verwerking kan worden bereikt met behulp van berichtenwachtrijen, event-driven architecturen of frameworks voor achtergrondtaakverwerking zoals Sidekiq.

2. **Gedistribueerde Architectuur**: Ontwerp de systeemarchitectuur om serverloze componenten (zoals AWS Lambda) te gebruiken of verdeel de werklast eenvoudig over meerdere nodes of servers naast je hoofdapplicatieserver. Dit maakt horizontale schaalbaarheid mogelijk, waarbij extra nodes kunnen worden toegevoegd om verhoogde workflowvolumes te verwerken.

3. **Parallelle Uitvoering**: Identificeer mogelijkheden voor parallelle uitvoering binnen workflows. Sommige workflowcomponenten kunnen onafhankelijk van elkaar zijn en gelijktijdig worden uitgevoerd. Door gebruik te maken van parallelle verwerkingstechnieken, zoals multi-threading of gedistribueerde taakwachtrijen, kan het systeem het resourcegebruik optimaliseren en de totale uitvoeringstijd van workflows verminderen.

Prestatie-optimalisatie van AI-gestuurde Componenten

AI-gestuurde componenten, zoals machine learning-modellen of natuurlijke taalverwerkingsmotoren, kunnen rekenintensief zijn en invloed hebben op de algehele prestaties van het workfloworchestratie-systeem. Overweeg de volgende technieken om de prestaties van AI-componenten te optimaliseren:

1. **Caching**: Als je AI-verwerking puur generatief is en geen realtime informatie-opzoekingen of externe integraties vereist om chat-voltooiingen te genereren, dan kun je caching-mechanismen onderzoeken om de resultaten van veelgebruikte of rekenintensieve bewerkingen op te slaan en te hergebruiken.

2. **Model-optimalisatie**: Optimaliseer voortdurend de manier waarop je AI-modellen gebruikt in workflowcomponenten. Dit kan technieken omvatten zoals *Prompt Distillatie* of het kan simpelweg een kwestie zijn van het testen van nieuwe modellen zodra deze beschikbaar komen.

3. **Batchverwerking**: Als je werkt met GPT-4-klasse modellen, kun je mogelijk gebruik maken van batchverwerkingstechnieken om meerdere datapunten of verzoeken in één batch te verwerken, in plaats van ze individueel te verwerken. Door gegevens in batches te verwerken, kan het systeem het brongebruik optimaliseren en de overhead van herhaalde modelverzoeken verminderen.

Monitoren en Profileren van Prestaties

Om prestatie-knelpunten te identificeren en de schaalbaarheid van het intelligente workfloworchestratie-systeem te optimaliseren, is het cruciaal om monitoring- en profileringsmechanismen te implementeren. Overweeg de volgende benaderingen:

1. **Prestatiemetrieken**: Definieer en volg belangrijke prestatiemetrieken, zoals responstijd, doorvoer, brongebruik en latentie. Deze metrieken geven inzicht in de systeemprestaties en helpen gebieden voor optimalisatie te identificeren. De populaire AI-model aggregator OpenRouter bevat Host[1] en Speed[2] metrieken in elke API-respons, waardoor het eenvoudig is om deze belangrijke metrieken te volgen.

2. **Profileringstools**: Gebruik profileringstools om de prestaties van individuele workflowcomponenten en AI-operaties te analyseren. Profileringstools kunnen helpen bij het identificeren van prestatie-hotspots, inefficiënte codepaden of bronintensieve operaties. Populaire profileringstools zijn onder andere New Relic, Scout, of ingebouwde profilers die worden geleverd bij de programmeertaal of het framework.

3. **Belastingstests**: Voer belastingstests uit om de systeemprestaties onder verschillende niveaus van gelijktijdige werklasten te evalueren. Belastingstests helpen bij het identificeren van de schaalbaarheidslimieten van het systeem, het detecteren van prestatieverslechtering en het waarborgen dat het systeem het verwachte verkeer kan verwerken zonder de prestaties in gevaar te brengen.

4. **Continue Monitoring**: Implementeer continue monitoring- en waarschuwingsmechanismen om proactief prestatieproblemen en knelpunten te detecteren. Stel monitoringdashboards en waarschuwingen in om belangrijke prestatie-indicatoren (KPI's) te volgen en meldingen te ontvangen wanneer vooraf gedefinieerde drempels worden overschreden. Dit maakt snelle identificatie en oplossing van prestatieproblemen mogelijk.

[1] Host is de tijd die nodig was om de eerste byte van de gestreamde generatie van de modelhost te ontvangen, ook wel bekend als "time to first byte."

[2] Speed wordt berekend als het aantal voltooiingstokens gedeeld door de totale generatietijd. Voor niet-gestreamde verzoeken wordt latentie beschouwd als onderdeel van de generatietijd.

Schalingsstrategieën

Overweeg de volgende schalingsstrategieën om toenemende werklasten te verwerken en de schaalbaarheid van het intelligente workfloworchestratie-systeem te waarborgen:

1. **Verticaal Schalen**: Verticaal schalen betreft het verhogen van de resources (bijv. CPU, geheugen) van individuele nodes of servers om hogere werklasten aan te kunnen. Deze aanpak is geschikt wanneer het systeem meer verwerkingskracht of geheugen nodig heeft om complexe workflows of AI-operaties uit te voeren.

2. **Horizontaal Schalen**: Horizontaal schalen betreft het toevoegen van meer nodes of servers aan het systeem om de werklast te verdelen. Deze aanpak is effectief wanneer het systeem een groot aantal gelijktijdige workflows moet verwerken of wanneer de werklast gemakkelijk over meerdere nodes kan worden verdeeld. Horizontaal schalen vereist een gedistribueerde architectuur en load balancing-mechanismen om een gelijkmatige verdeling van verkeer te waarborgen.

3. **Automatisch Schalen**: Implementeer automatische schalingsmechanismen om het aantal nodes of resources automatisch aan te passen op basis van de werklastvraag. Automatisch schalen stelt het systeem in staat om dynamisch op en af te schalen afhankelijk van het inkomende verkeer, wat zorgt voor optimaal brongebruik en kostenefficiëntie. Cloudplatforms zoals Amazon Web Services (AWS) of Google Cloud Platform (GCP) bieden automatische schalingsmogelijkheden die kunnen worden benut voor intelligente workfloworchestratie-systemen.

Prestatie-optimalisatietechnieken

Naast de schalingsstrategieën, overweeg de volgende prestatie-optimalisatietechnieken om de efficiëntie van het intelligente workfloworchestratie-systeem te verbeteren:

1. **Efficiënte Gegevensopslag en -ophaling**: Optimaliseer de mechanismen voor gegevensopslag en -ophaling die door de workflowcomponenten worden gebruikt. Gebruik efficiënte database-indexering, query-optimalisatietechnieken en

gegevenscaching om de latentie te minimaliseren en de prestaties van data-intensieve operaties te verbeteren.

2. **Asynchrone I/O**: Maak gebruik van asynchrone I/O-operaties om blokkering te voorkomen en de reactiesnelheid van het systeem te verbeteren. Asynchrone I/O stelt het systeem in staat om meerdere verzoeken gelijktijdig af te handelen zonder te wachten op de voltooiing van I/O-operaties, waardoor de bronnen optimaal worden benut.

3. **Efficiënte Serialisatie en Deserialisatie**: Optimaliseer de serialisatie- en deserialisatieprocessen die worden gebruikt voor gegevensuitwisseling tussen werkstroomcomponenten. Gebruik efficiënte serialisatieformaten, zoals Protocol Buffers of MessagePack, om de overhead van gegevensserialisatie te verminderen en de prestaties van communicatie tussen componenten te verbeteren.

Overweeg voor Ruby-gebaseerde applicaties het gebruik van Universal ID. Universal ID maakt gebruik van zowel MessagePack als Brotli (een combinatie gebouwd voor snelheid en beste-in-zijn-klasse datacompressie). In combinatie zijn deze bibliotheken tot 30% sneller en komen binnen 2-5% compressieratio's in vergelijking met Protocol Buffers.

4. **Compressie en Codering**: Pas compressie- en coderingstechnieken toe om de omvang van gegevensoverdracht tussen werkstroomcomponenten te verminderen. Compressiealgoritmen, zoals gzip of Brotli, kunnen het netwerkbandbreedtegebruik aanzienlijk verminderen en de algehele prestaties van het systeem verbeteren.

Door rekening te houden met schaalbaarheid en prestatieaspecten tijdens het ontwerp en de implementatie van intelligente werkstroomorganisatiesystemen, kunt u ervoor zorgen dat uw systeem grote hoeveelheden gelijktijdige werkstromen kan verwerken, de prestaties van AI-gestuurde componenten kan optimaliseren en naadloos kan schalen om aan groeiende eisen te voldoen. Continue monitoring, profilering en optimalisatie-inspanningen zijn essentieel om de prestaties en reactiesnelheid van het systeem te behouden naarmate de werklast en complexiteit in de loop van de tijd toenemen.

Testen en Validatie van Werkstromen

Testen en validatie zijn cruciale aspecten bij het ontwikkelen en onderhouden van intelligente werkstroomorganisatiesystemen. Gezien de complexe aard van AI-gestuurde werkstromen is het essentieel om ervoor te zorgen dat elke component naar verwachting functioneert, de algemene werkstroom correct verloopt en de AI-beslissingen nauwkeurig en betrouwbaar zijn. In deze sectie verkennen we verschillende technieken en overwegingen voor het testen en valideren van intelligente werkstromen.

Unit Testen van Werkstroomcomponenten

Unit testen omvat het testen van individuele werkstroomcomponenten in isolatie om hun correctheid en robuustheid te verifiëren. Houd bij het unit testen van AI-gestuurde werkstroomcomponenten rekening met het volgende:

1. **Invoervalidatie**: Test het vermogen van de component om verschillende soorten invoer te verwerken, inclusief geldige en ongeldige gegevens. Verifieer dat de component randgevallen correct afhandelt en passende foutmeldingen of uitzonderingen genereert.

2. **Uitvoerverificatie**: Controleer of de component de verwachte uitvoer produceert voor een gegeven set invoergegevens. Vergelijk de daadwerkelijke uitvoer met de verwachte resultaten om correctheid te waarborgen.

3. **Foutafhandeling**: Test de foutafhandelingsmechanismen van de component door verschillende foutscenario's te simuleren, zoals ongeldige invoer, onbeschikbaarheid van bronnen of onverwachte uitzonderingen. Verifieer dat de component fouten op de juiste manier opvangt en afhandelt.

4. **Randvoorwaarden**: Test het gedrag van de component onder randvoorwaarden, zoals lege invoer, maximale invoergrootte of extreme waarden. Zorg ervoor dat de component deze voorwaarden correct afhandelt zonder vast te lopen of onjuiste resultaten te produceren.

Hier is een voorbeeld van een unit test voor een werkstroomcomponent in Ruby met behulp van het RSpec testframework:

```
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
```

In dit voorbeeld wordt de `OrderValidator` component getest met behulp van twee testgevallen: één voor een geldige bestelling en een andere voor een ongeldige bestelling. De testgevallen verifiëren dat de `validate` methode de verwachte booleaanse waarde teruggeeft op basis van de geldigheid van de bestelling.

Integratietesten van Workflow-interacties

Integratietesten richt zich op het verifiëren van de interacties en datastromen tussen verschillende workflowcomponenten. Het zorgt ervoor dat de componenten naadloos samenwerken en de verwachte resultaten produceren. Bij het integratietesten van intelligente workflows moet je rekening houden met het volgende:

1. **Componentinteractie**: Test de communicatie en gegevensuitwisseling tussen

workflowcomponenten. Verifieer dat de output van één component correct wordt doorgegeven als input aan de volgende component in de workflow.

2. **Dataconsistentie**: Zorg ervoor dat gegevens consistent en nauwkeurig blijven terwijl ze door de workflow stromen. Verifieer dat gegevenstransformaties, berekeningen en aggregaties correct worden uitgevoerd.

3. **Exceptieverspreiding**: Test hoe uitzonderingen en fouten worden verspreid en afgehandeld tussen workflowcomponenten. Verifieer dat uitzonderingen worden opgevangen, gelogd en op de juiste manier worden afgehandeld om verstoring van de workflow te voorkomen.

4. **Asynchroon gedrag**: Als de workflow asynchrone componenten of parallelle uitvoering bevat, test dan de coördinatie- en synchronisatiemechanismen. Zorg ervoor dat de workflow correct functioneert in gelijktijdige en asynchrone scenario's.

Hier is een voorbeeld van een integratietest voor een workflow in Ruby met behulp van het RSpec testframework:

```
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
```

In dit voorbeeld wordt de `OrderProcessingWorkflow` getest door de interacties tussen verschillende workflow-componenten te verifiëren. De testcase stelt verwachtingen op voor het gedrag van elk component en zorgt ervoor dat de workflow de order succesvol verwerkt, waarbij de orderstatus dienovereenkomstig wordt bijgewerkt.

Het Testen van AI-Beslispunten

Het testen van AI-beslispunten is cruciaal om de nauwkeurigheid en betrouwbaarheid van AI-gestuurde workflows te waarborgen. Houd bij het testen van AI-beslispunten rekening met het volgende:

1. **Beslissingsnauwkeurigheid**: Verifieer dat het AI-component accurate beslissingen neemt op basis van de invoergegevens en het getrainde model. Vergelijk de AI-beslissingen met verwachte uitkomsten of referentiegegevens.

2. **Randgevallen**: Test het gedrag van het AI-component in randgevallen en ongebruikelijke scenario's. Verifieer dat het AI-component deze gevallen correct afhandelt en redelijke beslissingen neemt.

3. **Vooringenomenheid en Eerlijkheid**: Beoordeel het AI-component op mogelijke vooroordelen en zorg ervoor dat het eerlijke en onbevooroordeelde beslissingen neemt. Test het component met diverse invoergegevens en analyseer de uitkomsten op discriminerende patronen.

4. **Verklaarbaarheid**: Als het AI-component uitleg of redenering geeft voor zijn beslissingen, verifieer dan de juistheid en helderheid van de uitleg. Zorg ervoor dat de uitleg overeenkomt met het onderliggende besluitvormingsproces.

Hier is een voorbeeld van het testen van een AI-beslispunt in Ruby met behulp van het RSpec testraamwerk:

```
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
```

In dit voorbeeld wordt de `FraudDetector` AI-component getest met twee testgevallen: één voor een frauduleuze transactie en een andere voor een legitieme transactie. De testgevallen verifiëren of de `detect_fraud` methode de verwachte booleaanse waarde teruggeeft op basis van de kenmerken van de transactie.

End-to-End Testen

End-to-end testen omvat het testen van de volledige workflow van begin tot eind, waarbij realistische scenario's en gebruikersinteracties worden gesimuleerd. Het zorgt ervoor dat de workflow correct functioneert en de gewenste resultaten oplevert. Bij het uitvoeren van end-to-end testen voor intelligente workflows moet je rekening houden met het volgende:

1. **Gebruiksscenario's**: Identificeer veelvoorkomende gebruiksscenario's en test het gedrag van de workflow in deze scenario's. Verifieer dat de workflow correct omgaat met gebruikersinvoer, passende beslissingen neemt en de verwachte uitvoer produceert.

2. **Datavalidatie**: Zorg ervoor dat de workflow gebruikersinvoer valideert en opschoont om data-inconsistenties of beveiligingskwetsbaarheden te voorkomen. Test de workflow met verschillende soorten invoergegevens, waaronder zowel geldige als ongeldige data.

3. **Foutherstel**: Test het vermogen van de workflow om te herstellen van fouten en uitzonderingen. Simuleer foutscenario's en verifieer dat de workflow deze correct afhandelt, de fouten logt en passende herstelacties onderneemt.

4. **Prestaties en Schaalbaarheid**: Beoordeel de prestaties en schaalbaarheid van de workflow onder verschillende belastingsomstandigheden. Test de workflow met een groot volume aan gelijktijdige verzoeken en meet de responstijden, het resourcegebruik en de algemene systeemstabiliteit.

Hier is een voorbeeld van een end-to-end test voor een workflow in Ruby met behulp van het RSpec testframework en de Capybara bibliotheek voor het simuleren van gebruikersinteracties:

```
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
```

In dit voorbeeld simuleert de end-to-end test een gebruiker die een bestelling plaatst via de webinterface. Het vult de vereiste formuliervelden in, verstuurt de bestelling

en verifieert dat de bestelling succesvol wordt verwerkt, waarbij het de juiste bevestigingsboodschap toont en de bestelstatus in de database bijwerkt.

Continue Integratie en Implementatie

Om de betrouwbaarheid en onderhoudbaarheid van intelligente workflows te waarborgen, wordt aangeraden om testen en validatie te integreren in de continuous integration and deployment (CI/CD) pipeline. Dit maakt geautomatiseerd testen en valideren van workflow-wijzigingen mogelijk voordat ze in productie worden genomen. Houd rekening met de volgende praktijken:

1. **Geautomatiseerde Testuitvoering**: Configureer de CI/CD-pipeline om automatisch de testsuite uit te voeren wanneer er wijzigingen worden aangebracht in de workflow-codebase. Dit zorgt ervoor dat eventuele regressies of fouten vroeg in het ontwikkelingsproces worden ontdekt.

2. **Testdekkingsbewaking**: Meet en monitor de testdekking van de workflow-componenten en AI-beslispunten. Streef naar een hoge testdekking om ervoor te zorgen dat kritieke paden en scenario's grondig worden getest.

3. **Continue Feedback**: Integreer testresultaten en codekwaliteitsmetrieken in de ontwikkelworkflow. Voorzie ontwikkelaars van continue feedback over de status van tests, codekwaliteit en eventuele problemen die tijdens het CI/CD-proces worden gedetecteerd.

4. **Stagingomgevingen**: Implementeer de workflow in stagingomgevingen die de productieomgeving zo dicht mogelijk benaderen. Voer aanvullende tests en validatie uit in de stagingomgeving om eventuele problemen met infrastructuur, configuratie of data-integratie op te sporen.

5. **Terugrolmechanismen**: Implementeer terugrolmechanismen voor het geval er implementatiefouten of kritieke problemen in productie worden ontdekt. Zorg ervoor

dat de workflow snel kan worden teruggezet naar een vorige stabiele versie om downtime en impact op gebruikers te minimaliseren.

Door testen en validatie te integreren gedurende de hele ontwikkelingslevenscyclus van intelligente workflows, kunnen organisaties de betrouwbaarheid, nauwkeurigheid en onderhoudbaarheid van hun AI-gestuurde systemen waarborgen. Regelmatig testen en valideren helpt bij het opsporen van bugs, het voorkomen van regressies en het opbouwen van vertrouwen in het gedrag en de resultaten van de workflow.

Deel 2: De Patronen

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Prompt Engineering

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Chain of Thought

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Hoe het werkt

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Voorbeelden

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Content Generatie

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Gestructureerde Entiteitscreatie

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

LLM-Agent Begeleiding

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Voordelen en Overwegingen

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Mode Switch

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Hoe het Werkt

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Wanneer te gebruiken

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Voorbeeld

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Roltoewijzing

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Hoe Het Werkt

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Wanneer Te Gebruiken

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Voorbeelden

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Prompt Object

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Hoe Het Werkt

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Promptsjabloon

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Hoe het werkt

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Voordelen en Overwegingen

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Wanneer te gebruiken:

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Voorbeeld

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Structured IO

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Hoe het werkt

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Schalen van Structured IO

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Voordelen en Overwegingen

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Prompt Chaining

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Hoe Het Werkt

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Wanneer Te Gebruiken

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Voorbeeld: Olympia's Onboarding

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Prompt Rewriter

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Hoe Het Werkt

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Voorbeeld

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Responsbegrenzing

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Hoe het werkt

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Voordelen en Overwegingen

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Foutafhandeling

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Query Analyzer

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Hoe Het Werkt

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Implementatie

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Part-of-Speech (POS) Tagging en Named Entity Recognition (NER)

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Intentie-classificatie

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Trefwoordextractie

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Voordelen

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Query Rewriter

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Hoe Het Werkt

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Voorbeeld

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Voordelen

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Ventriloquist

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Hoe Het Werkt

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Wanneer Te Gebruiken

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Voorbeeld

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Discrete Componenten

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Predicaat

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Hoe Het Werkt

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Wanneer te gebruiken

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Voorbeeld

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

API-façade

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Hoe het werkt

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Belangrijkste voordelen

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Wanneer te gebruiken

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Voorbeeld

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Authenticatie en Autorisatie

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Verwerking van Verzoeken

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Opmaak van Antwoorden

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Foutafhandeling en Randgevallen

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Overwegingen voor Schaalbaarheid en Prestaties

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Vergelijking met Andere Ontwerppatronen

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Resultaatinterpreteerder

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Hoe Het Werkt

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Wanneer Te Gebruiken

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Voorbeeld

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Virtuele Machine

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Hoe Het Werkt

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Wanneer Te Gebruiken

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Voorbeeld

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Achter De Magie

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Specificatie en Testen

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Het Specificeren van het Gedrag

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Testgevallen Schrijven

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Voorbeeld: Het Testen van de Vertaler Component

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Afspelen van HTTP-interacties

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Human In The Loop (HITL)

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Patronen op Hoog Niveau

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Hybride Intelligentie

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Adaptieve Respons

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Mens-AI Rolverwisseling

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Escalatie

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Hoe Het Werkt

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Belangrijkste Voordelen

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Praktijktoepassing: Gezondheidszorg

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Feedbackloop

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Hoe het werkt

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Toepassingen en Voorbeelden

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Geavanceerde Technieken in Menselijke Feedback-integratie

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Passieve Informatie-uitstraling

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Hoe Het Werkt

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Contextuele Informatieweergave

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Proactieve Meldingen

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Verklarende Inzichten

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Interactieve Verkenning

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Belangrijke Voordelen

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Toepassingen en Voorbeelden

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Gezamenlijke Besluitvorming (CDM)

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Hoe Het Werkt

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Voorbeeld

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Continue Learning

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Hoe het werkt

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Toepassingen en Voorbeelden

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Voorbeeld

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Ethische Overwegingen

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Rol van HITL bij het Beperken van AI-risico's

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Technologische Vooruitgang en Toekomstperspectief

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Uitdagingen en Beperkingen van HITL-systemen

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Intelligente Foutafhandeling

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Traditionele Foutafhandelingsbenaderingen

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Contextuele Foutdiagnose

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Hoe het Werkt

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Prompt Engineering voor Contextuele Foutdiagnose

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Retrieval-Augmented Generation voor Contextuele Foutdiagnose

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Intelligente Foutrapportage

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Voorspellende Foutpreventie

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Hoe Het Werkt

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Slim Foutherstel

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Hoe Het Werkt

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Gepersonaliseerde Foutcommunicatie

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Hoe Het Werkt

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Adaptieve Foutafhandelingsworkflow

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Hoe het werkt

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Kwaliteitscontrole

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Eval

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Probleem

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Oplossing

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Hoe Het Werkt

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Voorbeeld

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Overwegingen

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Begrip van Gouden Referenties

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Hoe Referentievrije Evaluaties Werken

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Veiligheidsrail

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Probleem

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Oplossing

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Hoe het werkt

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Voorbeeld

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Overwegingen

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Guardrails en Evals: Twee Kanten van Dezelfde Medaille

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

De Uitwisselbaarheid van Guardrails en Referentievrije Evals

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Implementatie van Dual-Purpose Guardrails en Evals

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Begrippenlijst

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Begrippenlijst

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

A

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

B

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

C

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

D

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

E

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

F

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

G

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

H

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

I

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

J

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

K

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

L

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

M

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

N

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

O

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

P

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Q

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

R

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

S

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

T

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

U

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

V

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

W

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Z

Deze inhoud is niet beschikbaar in het sample boek. Het boek kan aangekocht worden op deze Leanpub link: http://leanpub.com/patterns-of-application-development-using-ai-nl.

Index