



Patterns of Application Development Using AI

Obie Fernandez
Foreword by Gregor Hohpe



Leanpub

Deutsche Ausgabe

Muster der Anwendungsentwicklung mit KI (Deutsche Ausgabe)

Obie Fernandez

Dieses Buch wird verkauft unter

<http://leanpub.com/patterns-of-application-development-using-ai-de>

Diese Version wurde veröffentlicht am 2025-01-23



Dies ist ein [Leanpub](#)-Buch. Leanpub bietet Autoren und Verlagen, mit Hilfe von Lean-Publishing, neue Möglichkeiten des Publizierens. [Lean Publishing](#) bedeutet die wiederholte Veröffentlichung neuer Beta-Versionen eines eBooks unter der Zuhilfenahme schlanker Werkzeuge. Das Feedback der Erstleser hilft dem Autor bei der Finalisierung und der anschließenden Vermarktung des Buches. Lean Publishing unterstützt den Autor darin ein Buch zu schreiben, das auch gelesen wird.

© 2025 Obie Fernandez

Twitterte dieses Buch!

Bitte unterstütze Obie Fernandez, indem du dieses Buch auf [Twitter](#) weiterempfehlst!

Vorschlag: Verwende den folgenden Hashtag, wenn du über dieses Buch twitterst:

[#poaduai](#).

Was sagen andere über dieses Buch? Klicke hier, um nach diesem Hashtag auf Twitter zu suchen:

[#poaduai](#)

Für meine unerschrockene Königin, meine Muse, mein Licht und meine Liebe, Victoria

Ebenfalls von 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

Inhaltsverzeichnis

Vorwort von Gregor Hohpe	i
Vorwort	ii
Über das Buch	iii
Über die Codebeispiele	iii
Was ich nicht behandle	iii
Für wen ist dieses Buch gedacht	iii
Aufbau eines gemeinsamen Vokabulars	iii
Mitmachen	iii
Danksagungen	iii
Was hat es mit den Illustrationen auf sich?	iv
Über Lean Publishing	iv
Über den Autor	v
Einführung	1
Gedanken zur Softwarearchitektur	2
Was ist ein Large Language Model?	3
Inferenz verstehen	5
Überlegungen zur Leistung	28
Experimentieren mit verschiedenen LLM-Modellen	30
Zusammengesetzte KI-Systeme	31

Teil 1: Grundlegende Ansätze & Techniken . . 39

Den Pfad eingrenzen	40
Latenter Raum: Unfassbar weit	42
Wie der Pfad “eingegrenzt” wird	47
Rohmodelle versus instruktionsoptimierte Modelle	51
Prompt-Engineering	58
Prompt Distillation	76
Was ist mit Fine-tuning?	83
Retrieval Augmented Generation (RAG)	84
Was ist Retrieval Augmented Generation?	84
Wie funktioniert RAG?	84
Warum RAG in Ihren Anwendungen einsetzen?	84
Implementierung von RAG in Ihrer Anwendung	84
Propositions-Chunking	85
Praxisbeispiele für RAG	85
Intelligente Abfrageoptimierung (IQO)	86
Neuordnung	86
RAG-Bewertung (RAGAs)	86
Herausforderungen und Zukunftsausblick	88
Vielzahl von Arbeitern	90
KI-Arbeiter als unabhängige wiederverwendbare Komponenten	91
Kontoverwaltung	93
E-Commerce-Anwendungen	94
Anwendungen im Gesundheitswesen	103
KI-Worker als Prozess-Manager	107
Integration von KI-Workern in Ihre Anwendungsarchitektur	111
Komponierbarkeit und Orchestrierung von KI-Workern	115

INHALTSVERZEICHNIS

Kombination von traditionellem NLP mit LLMs	125
Werkzeugnutzung	128
Was ist Werkzeugnutzung?	128
Das Potenzial des Werkzeugeinsatzes	130
Der Werkzeugeinsatz-Workflow	131
Best Practices für die Werkzeugnutzung	147
Zusammenstellung und Verkettung von Werkzeugen	152
Zukünftige Entwicklungen	154
Stream Processing	157
Implementierung eines ReplyStream	158
Die “Konversationsschleife”	164
Automatische Fortsetzung	167
Fazit	169
Selbstheilende Daten	171
Praktische Fallstudie: Reparatur von fehlerhaftem JSON	174
Überlegungen und Gegenanzeigen	179
Kontextbezogene Inhaltsgenerierung	195
Personalisierung	196
Produktivität	198
Schnelle Iteration und Experimentierung	201
KI-gestützte Lokalisierung	204
Die Bedeutung von Benutzertests und Feedback	206
Generative UI	207
Generierung von Texten für Benutzeroberflächen	209
Definition der Generativen UI	218
Beispiel	220

Der Wandel zum ergebnisorientierten Design	223
Herausforderungen und Überlegungen	224
Zukunftsausblick und Chancen	226
Intelligente Workflow-Orchestrierung	230
Geschäftlicher Bedarf	231
Wichtige Vorteile	232
Wichtige Muster	233
Ausnahmebehandlung und Wiederherstellung	235
Implementierung der intelligenten Workflow-Orchestrierung in der Praxis	239
Überwachung und Protokollierung	253
Skalierbarkeits- und Leistungsüberlegungen	259
Testen und Validierung von Workflows	264
 Teil 2: Die Muster	 272
Prompt-Engineering	273
Chain of Thought	274
Moduswechsel	275
Rollenzuweisung	276
Prompt-Objekt	277
Prompt-Vorlage	278
Structured IO	279
Prompt Chaining	280
Prompt Rewriter	281
Response Fencing	282
Query-Analyzer	283
Query Rewriter	284
Ventriloquist	285

Diskrete Komponenten	286
Prädikat	287
API-Fassade	288
Result Interpreter	290
Virtuelle Maschine	291
Spezifikation und Testing	291
Human In The Loop (HITL)	293
Übergeordnete Muster	293
Eskalation	294
Feedback-Schleife	295
Passive Informationsabstrahlung	296
Kollaborative Entscheidungsfindung (CDM)	298
Kontinuierliches Lernen	299
Ethische Überlegungen	299
Technologische Fortschritte und Zukunftsausblick	299
Intelligente Fehlerbehandlung	301
Traditionelle Fehlerbehandlungsansätze	301
Kontextuelle Fehlerdiagnose	302
Intelligente Fehlerberichterstattung	303
Prädiktive Fehlervermeidung	304
Intelligente Fehlerbehebung	304
Personalisierte Fehlerkommunikation	305
Adaptiver Fehlerbehandlungs-Workflow	306
Qualitätskontrolle	307
Eval	308
Schutzmechanismus	310
Guardrails und Evaluierungen: Zwei Seiten derselben Medaille	310

Glossar 312
 Glossar 312
Register 317

Vorwort von Gregor Hohpe

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Vorwort

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Über das Buch

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Über die Codebeispiele

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Was ich nicht behandle

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Für wen ist dieses Buch gedacht

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Aufbau eines gemeinsamen Vokabulars

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Mitmachen

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Danksagungen

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Was hat es mit den Illustrationen auf sich?

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

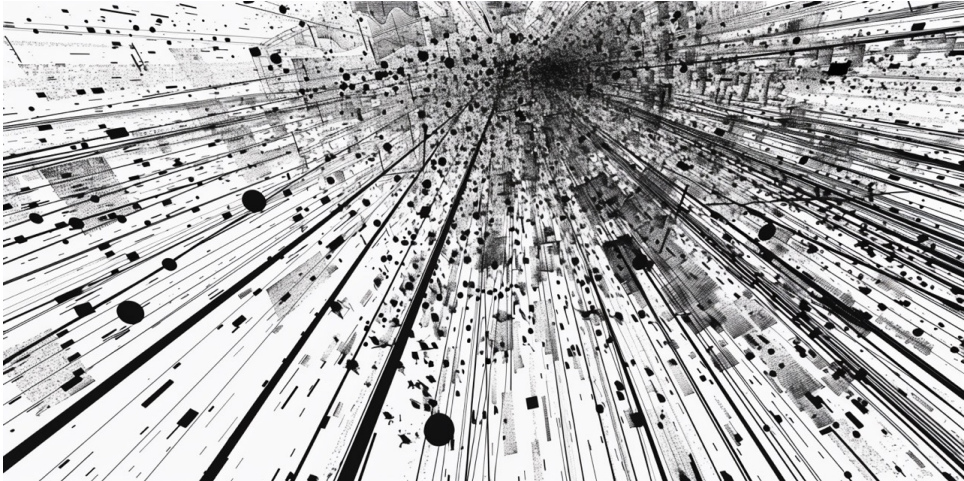
Über Lean Publishing

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Über den Autor

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Einführung



Wenn Sie begierig darauf sind, KI-gestützte Large Language Models (LLMs) in Ihre Programmierprojekte zu integrieren, können Sie direkt mit den Mustern und Codebeispielen in den späteren Kapiteln beginnen. Um jedoch die Kraft und das Potenzial dieser Muster vollständig zu verstehen, lohnt es sich, einen Moment innezuhalten und den breiteren Kontext sowie den kohärenten Ansatz zu erfassen, den sie darstellen.

Die Muster sind nicht nur eine Sammlung isolierter Techniken, sondern vielmehr ein einheitliches Framework für die Integration von KI in Ihre Anwendungen. Ich verwende Ruby on Rails, aber diese Muster sollten in praktisch jeder anderen Programmierumgebung funktionieren. Sie behandeln ein breites Spektrum von Aspekten, von Datenverwaltung und Leistungsoptimierung bis hin zu Benutzererfahrung und Sicherheit, und bieten damit ein umfassendes Werkzeug zur Erweiterung traditioneller Programmierpraktiken mit KI-Fähigkeiten.

Jede Kategorie von Mustern geht eine spezifische Herausforderung oder Chance an, die sich bei der Integration von KI-Komponenten in Ihre Anwendung ergibt. Durch das Verständnis der Beziehungen und Synergien zwischen diesen Mustern können Sie fundierte Entscheidungen darüber treffen, wo und wie Sie KI am effektivsten einsetzen können.

Muster sind niemals präskriptive Lösungen und sollten auch nicht als solche behandelt werden. Sie sind als anpassbare Bausteine gedacht, die auf die einzigartigen Anforderungen und Einschränkungen Ihrer eigenen Anwendung zugeschnitten werden sollten. Die erfolgreiche Anwendung dieser Muster (wie bei allen anderen im Softwarebereich) basiert auf einem tiefen Verständnis der Problemdomäne, der Benutzerbedürfnisse und der gesamten technischen Architektur Ihres Projekts.

Gedanken zur Softwarearchitektur

Ich begann in den 1980er Jahren mit dem Programmieren und war in der Hackerszene aktiv. Diese Hacker-Denkweise habe ich auch nach meinem Einstieg in die professionelle Softwareentwicklung nie verloren. Von Anfang an hatte ich eine gesunde Skepsis gegenüber dem tatsächlichen Mehrwert, den Softwarearchitekten in ihren Elfenbeintürmen lieferten.

Einer der Gründe, warum ich persönlich so begeistert von den Veränderungen bin, die diese neue, leistungsstarke KI-Technologie mit sich bringt, ist ihre Auswirkung auf das, was wir als *Softwarearchitektur*-Entscheidungen betrachten. Sie stellt traditionelle Vorstellungen darüber in Frage, was die "richtige" Art ist, unsere Softwareprojekte zu entwerfen und umzusetzen. Sie stellt auch in Frage, ob Architektur noch primär als *die Teile eines Systems, die schwer zu ändern sind* betrachtet werden kann, da KI-Erweiterungen es einfacher als je zuvor machen, jeden Teil Ihres Projekts jederzeit zu ändern.

Vielleicht betreten wir gerade die Hochphase des "postmodernen" Ansatzes in der Softwareentwicklung. In diesem Kontext bezieht sich postmodern auf eine

fundamentale Abkehr von traditionellen Paradigmen, bei denen Entwickler für das Schreiben und Warten jeder Codezeile verantwortlich waren. Stattdessen wird die Idee der Delegation von Aufgaben wie Datenmanipulation, komplexen Algorithmen und sogar ganzen Teilen der Anwendungslogik an Drittanbieter-Bibliotheken und externe APIs befürwortet. Diese postmoderne Verschiebung stellt einen bedeutenden Abschied von der konventionellen Weisheit dar, Anwendungen von Grund auf neu zu entwickeln, und fordert Entwickler heraus, ihre Rolle im Entwicklungsprozess zu überdenken.

Ich habe immer geglaubt, dass gute Programmierer nur den Code schreiben, der absolut notwendig ist, basierend auf den Lehren von Larry Wall und anderen Hacker-Koryphäen wie ihm. Durch die Minimierung der Menge des geschriebenen Codes können wir schneller arbeiten, die Angriffsfläche für Fehler reduzieren, die Wartung vereinfachen und die allgemeine Zuverlässigkeit ihrer Anwendungen verbessern. Weniger Code ermöglicht es uns, uns auf die zentrale Geschäftslogik und Benutzererfahrung zu konzentrieren, während andere Arbeiten an andere Dienste delegiert werden.

Jetzt, da KI-gestützte Systeme Aufgaben übernehmen können, die früher ausschließlich von Menschen geschriebenem Code vorbehalten waren, sollten wir noch produktiver und agiler sein können, mit einem größeren Fokus denn je auf die Schaffung von Geschäftswert und Benutzererfahrung.

Natürlich gibt es auch Kompromisse bei der Delegation großer Teile Ihres Projekts an KI-Systeme, wie den potenziellen Kontrollverlust und die Notwendigkeit robuster Überwachungs- und Feedback-Mechanismen. Deshalb erfordert es neue Fähigkeiten und Kenntnisse, einschließlich zumindest eines grundlegenden Verständnisses davon, wie KI funktioniert.

Was ist ein Large Language Model?

Large Language Models (LLMs) sind eine Art künstlicher Intelligenz, die seit der Einführung von GPT-3 durch OpenAI im Jahr 2020 erhebliche Aufmerksamkeit

erregt haben. LLMs sind darauf ausgelegt, menschliche Sprache mit bemerkenswerter Genauigkeit und Flüssigkeit zu verarbeiten, zu verstehen und zu generieren. In diesem Abschnitt werfen wir einen kurzen Blick darauf, wie LLMs funktionieren und warum sie sich gut für den Aufbau intelligenter Systemkomponenten eignen.

Im Kern basieren LLMs auf Deep-Learning-Algorithmen, speziell auf neuronalen Netzen. Diese Netzwerke bestehen aus miteinander verbundenen Knoten oder Neuronen, die Informationen verarbeiten und übertragen. Die Architektur der Wahl für LLMs ist oft das Transformer-Modell, das sich als hocheffektiv bei der Verarbeitung sequentieller Daten wie Text erwiesen hat.

Transformer-Modelle basieren auf dem Aufmerksamkeitsmechanismus und werden hauptsächlich für Aufgaben mit sequenziellen Daten wie der Verarbeitung natürlicher Sprache eingesetzt. Transformer verarbeiten Eingabedaten gleichzeitig statt sequenziell, wodurch sie weitreichende Abhängigkeiten effektiver erfassen können. Sie verfügen über Schichten von Aufmerksamkeitsmechanismen, die dem Modell helfen, sich auf verschiedene Teile der Eingabedaten zu konzentrieren, um Kontext und Beziehungen zu verstehen.

Der Trainingsprozess für LLMs beinhaltet die Konfrontation des Modells mit riesigen Mengen an Textdaten, wie Bücher, Artikel, Websites und Code-Repositories. Während des Trainings lernt das Modell, Muster, Beziehungen und Strukturen innerhalb des Textes zu erkennen. Es erfasst die statistischen Eigenschaften der Sprache, wie Grammatikregeln, Wortassoziationen und kontextuelle Bedeutungen.

Eine der wichtigsten Techniken beim Training von LLMs ist das unüberwachte Lernen. Dies bedeutet, dass das Modell ohne explizite Kennzeichnung oder Anleitung aus den Daten lernt. Es entdeckt selbstständig Muster und Darstellungen, indem es das gemeinsame Auftreten von Wörtern und Phrasen in den Trainingsdaten analysiert. Dies ermöglicht es LLMs, ein tiefes Verständnis von Sprache und ihren Feinheiten zu entwickeln.

Ein weiterer wichtiger Aspekt von LLMs ist ihre Fähigkeit, *Kontext* zu verarbeiten. Bei

der Verarbeitung eines Textes berücksichtigen LLMs nicht nur die einzelnen Wörter, sondern auch den umgebenden Kontext. Sie berücksichtigen die vorherigen Wörter, Sätze und sogar Absätze, um die Bedeutung und Absicht des Textes zu verstehen. Dieses kontextuelle Verständnis ermöglicht es LLMs, kohärente und relevante Antworten zu generieren. Eine der wichtigsten Methoden zur Bewertung der Fähigkeiten eines bestimmten LLM-Modells ist die Betrachtung der Kontextgröße, die sie bei der Generierung von Antworten berücksichtigen können.

Nach dem Training können LLMs für ein breites Spektrum von sprachbezogenen Aufgaben eingesetzt werden. Sie können menschenähnlichen Text generieren, Fragen beantworten, Dokumente zusammenfassen, Sprachen übersetzen und sogar Code schreiben. Die Vielseitigkeit von LLMs macht sie wertvoll für den Aufbau intelligenter Systemkomponenten, die mit Benutzern interagieren, Textdaten verarbeiten und analysieren sowie aussagekräftige Ausgaben generieren können.

Durch die Integration von LLMs in die Anwendungsarchitektur können Sie KI-Komponenten erstellen, die Benutzereingaben verstehen und verarbeiten, dynamische Inhalte generieren und intelligente Empfehlungen oder Aktionen bereitstellen. Die Arbeit mit LLMs erfordert jedoch eine sorgfältige Berücksichtigung der Ressourcenanforderungen und Leistungskompromisse. LLMs sind rechenintensiv und können erhebliche Rechenleistung und Speicher (mit anderen Worten, Geld) für den Betrieb erfordern. Die meisten von uns müssen die Kostenauswirkungen der Integration von LLMs in unsere Anwendungen bewerten und entsprechend handeln.

Inferenz verstehen

Inferenz bezieht sich auf den Prozess, bei dem ein Modell Vorhersagen oder Ausgaben basierend auf neuen, ungesehenen Daten generiert. Es ist die Phase, in der das trainierte Modell verwendet wird, um Entscheidungen zu treffen oder Text, Bilder oder andere Inhalte als Reaktion auf Benutzereingaben zu generieren.

Während der Trainingsphase lernt ein KI-Modell aus einem großen Datensatz, indem es seine Parameter anpasst, um den Fehler in seinen Vorhersagen zu minimieren. Nach dem Training kann das Modell das Gelernte auf neue Daten anwenden. Inferenz ist die Art und Weise, wie das Modell seine gelernten Muster und Kenntnisse zur Generierung von Ausgaben nutzt.

Für LLMs beinhaltet die Inferenz die Aufnahme eines Prompts oder Eingabetextes und die Erzeugung einer kohärenten und kontextuell relevanten Antwort als Strom von *Token*(über die wir bald sprechen werden). Dies könnte die Beantwortung einer Frage, die Vervollständigung eines Satzes, die Generierung einer Geschichte oder die Übersetzung von Text sein, neben vielen anderen Aufgaben.



Im Gegensatz zu der Art und Weise, wie Sie und ich denken, geschieht das “Denken” eines KI-Modells durch Inferenz in einer einzigen zustandslosen Operation. Das heißt, sein Denken ist auf seinen Generierungsprozess beschränkt. Es muss buchstäblich laut denken, als ob ich Ihnen eine Frage stelle und nur eine Antwort von Ihnen im “Bewusstseinsstrom”-Stil akzeptiere.

Große Sprachmodelle gibt es in vielen Größen und Varianten

Während praktisch alle populären großen Sprachmodelle (LLMs) auf der gleichen Kern-Transformer-Architektur basieren und auf riesigen Textdatensätzen trainiert sind, kommen sie in verschiedenen Größen und sind für unterschiedliche Zwecke feinabgestimmt. Die Größe eines LLM, gemessen an der Anzahl der Parameter in seinem neuronalen Netzwerk, hat einen großen Einfluss auf seine Fähigkeiten. Größere Modelle mit mehr Parametern, wie GPT-4, das Gerüchten zufolge 1 bis 2 Billionen Parameter aufweist, sind im Allgemeinen kenntnisreicher und leistungsfähiger als kleinere Modelle. Allerdings benötigen größere Modelle auch viel mehr Rechenleistung, was sich in höheren Kosten bei der Nutzung über API-Aufrufe niederschlägt.

Um LLMs praktischer und für spezifische Anwendungsfälle maßgeschneiderter zu machen, werden die Basismodelle oft auf gezielteren Datensätzen feinabgestimmt. Zum Beispiel kann ein LLM auf einem großen Korpus von Dialogen trainiert werden, um es für konversationelle KI zu spezialisieren. Andere werden [auf Code trainiert](#), um ihnen Programmierkenntnisse zu vermitteln. Es gibt sogar Modelle, die [speziell für Rollenspiel-artige Interaktionen mit Benutzern trainiert wurden](#)!

Abrufbasierte vs. Generative Modelle

In der Welt der Large Language Models (LLMs) gibt es zwei hauptsächliche Ansätze zur Generierung von Antworten: abrufbasierte Modelle und generative Modelle. Jeder Ansatz hat seine eigenen Stärken und Schwächen, und das Verständnis der Unterschiede zwischen ihnen kann Ihnen helfen, das richtige Modell für Ihren spezifischen Anwendungsfall zu wählen.

Abrufbasierte Modelle

Abrufbasierte Modelle, auch bekannt als Informationsabruf-Modelle, generieren Antworten, indem sie eine große Datenbank mit bereits existierenden Texten durchsuchen und die relevantesten Passagen basierend auf der Eingabeanfrage auswählen. Diese Modelle generieren keine neuen Texte von Grund auf, sondern fügen stattdessen Auszüge aus der Datenbank zu einer kohärenten Antwort zusammen.

Einer der Hauptvorteile von abrufbasierten Modellen ist ihre Fähigkeit, sachlich korrekte und aktuelle Informationen bereitzustellen. Da sie sich auf eine Datenbank mit kuratierten Texten stützen, können sie relevante Informationen aus zuverlässigen Quellen abrufen und dem Benutzer präsentieren. Dies macht sie besonders geeignet für Anwendungen, die präzise, faktische Antworten erfordern, wie Frage-Antwort-Systeme oder Wissensdatenbanken.

Allerdings haben abrufbasierte Modelle auch einige Einschränkungen. Sie sind nur so gut wie die Datenbank, in der sie suchen, sodass die Qualität und Abdeckung der

Datenbank die Leistung des Modells direkt beeinflusst. Zusätzlich können diese Modelle Schwierigkeiten haben, kohärente und natürlich klingende Antworten zu generieren, da sie auf den in der Datenbank verfügbaren Text beschränkt sind.

In diesem Buch behandeln wir die Verwendung reiner Abrufmodelle nicht.

Generative Modelle

Generative Modelle hingegen erstellen neue Texte von Grund auf, basierend auf den Mustern und Beziehungen, die sie während des Trainings gelernt haben. Diese Modelle nutzen ihr Sprachverständnis, um neue Antworten zu generieren, die auf die Eingabeaufforderung zugeschnitten sind.

Die Hauptstärke generativer Modelle liegt in ihrer Fähigkeit, kreative, kohärente und kontextuell relevante Texte zu produzieren. Sie können offene Gespräche führen, Geschichten generieren und sogar Code schreiben. Dies macht sie ideal für Anwendungen, die offenere und dynamischere Interaktionen erfordern, wie Chatbots, Content-Erstellung und kreative Schreibassistenten.

Allerdings können generative Modelle manchmal inkonsistente oder sachlich falsche Informationen produzieren, da sie sich eher auf die während des Trainings gelernten Muster verlassen als auf eine kuratierte Faktendatenbank. Sie können auch anfälliger für Voreingenommenheit und Halluzinationen sein und Texte generieren, die plausibel erscheinen, aber nicht unbedingt wahr sind.

Beispiele für generative LLMs sind OpenAIs GPT-Serie (GPT-3, GPT-4) und Anthropic Claude.

Hybride Modelle

Mehrere kommerziell verfügbare LLMs kombinieren beide Ansätze – Abruf und Generierung – in einem hybriden Modell. Diese Modelle verwenden Abruftechniken, um relevante Informationen aus einer Datenbank zu finden, und nutzen dann generative Techniken, um diese Informationen zu einer kohärenten Antwort zu synthetisieren.

Hybride Modelle zielen darauf ab, die sachliche Genauigkeit von abrufbasierten Modellen mit den Fähigkeiten der natürlichen Sprachgenerierung von generativen Modellen zu verbinden. Sie können zuverlässigere und aktuellere Informationen bereitstellen und gleichzeitig die Fähigkeit zu offenen Gesprächen beibehalten.

Bei der Wahl zwischen abrufbasierten und generativen Modellen sollten Sie die spezifischen Anforderungen Ihrer Anwendung berücksichtigen. Wenn das primäre Ziel darin besteht, genaue, faktische Informationen bereitzustellen, könnte ein abrufbasiertes Modell die beste Wahl sein. Wenn die Anwendung offenere und kreativere Interaktionen erfordert, könnte ein generatives Modell besser geeignet sein. Hybride Modelle bieten ein Gleichgewicht zwischen den beiden Ansätzen und können eine gute Wahl für Anwendungen sein, die sowohl faktische Genauigkeit als auch natürliche Sprachgenerierung erfordern.

Letztendlich hängt die Wahl zwischen abrufbasierten und generativen Modellen vom spezifischen Anwendungsfall und den Kompromissen zwischen Genauigkeit, Kreativität und Flexibilität ab. Durch das Verständnis der Stärken und Einschränkungen jedes Ansatzes können Sie fundierte Entscheidungen treffen.

Instruktions-Feinabstimmung

Instruktions-Feinabstimmung ist eine Unterkategorie des Konversationstrainings, die menschlich geschriebene Prompts und Antworten verwendet, um LLMs besser darin zu machen, Anweisungen zu verstehen und zu befolgen. Hier sind einige einfache Beispiele, die veranschaulichen, wie Instruktionen-Feinabstimmung mit menschlich geschriebenen Prompts und Antworten LLMs besser darin macht, Anweisungen zu verstehen und zu befolgen:

Beispiel 1: Kundenservice

Anweisung: “Geben Sie eine Antwort an einen Kunden, der um eine Rückerstattung bittet.”

Menschlich geschriebener Prompt: “Ich möchte eine Rückerstattung für meinen kürzlichen Einkauf beantragen. Das Produkt hat meine Erwartungen nicht erfüllt.”

Menschlich geschriebene Antwort: “Es tut mir leid zu hören, dass das Produkt Ihre Erwartungen nicht erfüllt hat. Um Ihre Rückerstattung zu bearbeiten, geben Sie bitte Ihre Bestellnummer und den Grund für die Rücksendung an. Wir werden Ihre Anfrage prüfen und uns innerhalb von 2-3 Werktagen bei Ihnen melden.”

Beispiel 2: Bildungsunterstützung

Anweisung: “Erklären Sie das Konzept der Photosynthese einem 10-Jährigen.”

Von Menschen geschriebene Anfrage: “Kannst du erklären, wie Pflanzen ihre Nahrung herstellen?”

Von Menschen geschriebene Antwort: “Natürlich! Pflanzen stellen ihre Nahrung durch einen Prozess namens Photosynthese her. Sie nutzen Sonnenlicht, Wasser und Kohlendioxid, um Nahrung und Sauerstoff zu erzeugen. Die Blätter der Pflanze

haben einen speziellen Bestandteil namens Chlorophyll, der ihnen hilft, Sonnenlicht einzufangen.”

Beispiel 3: Persönlicher Assistent

Anweisung: “Plane ein Meeting für nächsten Dienstag um 15 Uhr.”

Von Menschen geschriebene Anfrage: “Kannst du ein Meeting für Dienstag um 15 Uhr einrichten?”

Von Menschen geschriebene Antwort: “Natürlich, ich habe ein Meeting für Sie am nächsten Dienstag um 15 Uhr geplant. Benötigen Sie sonst noch etwas?”

Das Ergebnis ist ein vielfältiges Ökosystem von LLMs in verschiedenen Größen und mit unterschiedlichen Spezialisierungen. Kleinere Modelle im Bereich von 1-7 Milliarden Parametern bieten gute allgemeine Sprachfähigkeiten bei gleichzeitig höherer Effizienz im Betrieb.

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

Mittelgroße Modelle mit etwa 30-70 Milliarden Parametern bieten stärkere Fähigkeiten im Bereich Reasoning und Befolgung von Anweisungen.

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

Bei der Auswahl eines LLM für eine Anwendung müssen die Fähigkeiten des Modells gegen praktische Faktoren wie Kosten, Latenz, Kontextlänge und Inhaltsfilterung

abgewogen werden. Kleinere, instruktionsoptimierte Modelle sind oft die beste Wahl für einfachere Sprachaufgaben, während die größten Modelle für komplexes Reasoning oder Analysen erforderlich sein können. Die Trainingsdaten des Modells sind ebenfalls ein wichtiger Aspekt, da sie den Wissens-Stichtag des Modells bestimmen.



Bestimmte Modelle, wie einige von Perplexity, sind mit Echtzeit-Informationsquellen verbunden, sodass sie effektiv keinen Stichtag haben. Wenn man ihnen Fragen stellt, können sie selbstständig entscheiden, Websuchen durchzuführen und beliebige Webseiten abzurufen, um eine Antwort zu generieren.

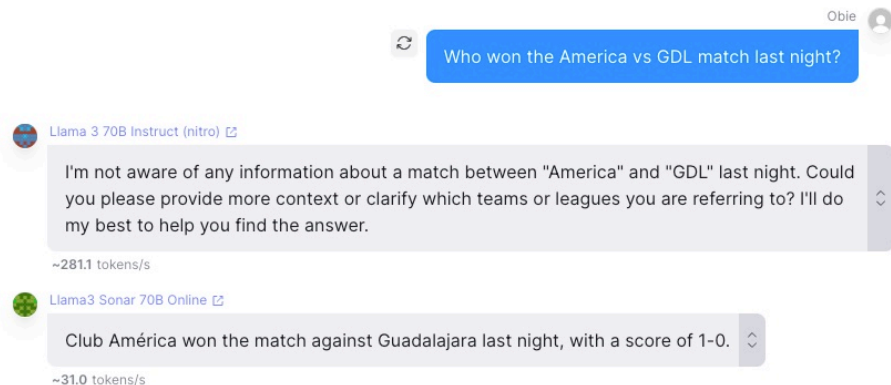
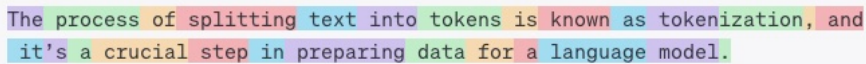


Abbildung 1. Llama3 mit und ohne Online-Zugang

Letztendlich gibt es kein universell einsetzbares LLM. Das Verständnis der Unterschiede in Modellgröße, Architektur und Training ist entscheidend für die Auswahl des richtigen Modells für einen bestimmten Anwendungsfall. Nur durch das Experimentieren mit verschiedenen Modellen lässt sich in der Praxis herausfinden, welche die beste Leistung für die jeweilige Aufgabe bieten.

Tokenisierung: Text in Stücke zerlegen

Bevor ein Large Language Model Text verarbeiten kann, muss dieser Text in kleinere Einheiten, sogenannte *Tokens*, zerlegt werden. Tokens können einzelne Wörter, Wortteile oder sogar einzelne Zeichen sein. Der Prozess der Zerlegung von Text in Tokens wird als Tokenisierung bezeichnet und ist ein entscheidender Schritt bei der Vorbereitung von Daten für ein Sprachmodell.



The process of splitting text into tokens is known as tokenization, and it's a crucial step in preparing data for a language model.

Abbildung 2. Dieser Satz enthält 27 Tokens

Verschiedene LLMs verwenden unterschiedliche Tokenisierungsstrategien, die einen erheblichen Einfluss auf die Leistung und Fähigkeiten des Modells haben können. Einige gängige Tokenizer, die von LLMs verwendet werden, sind:

- **GPT (Byte Pair Encoding):** GPT-Tokenizer verwenden eine Technik namens Byte Pair Encoding (BPE), um Text in Teilworteinheiten zu zerlegen. BPE verschmilzt iterativ die häufigsten Byte-Paare in einem Textkorpus und bildet so ein Vokabular aus Teilwort-Tokens. Dies ermöglicht es dem Tokenizer, seltene und neue Wörter zu verarbeiten, indem er sie in häufigere Teilwörter zerlegt. GPT-Tokenizer werden von Modellen wie GPT-3 und GPT-4 verwendet.
- **Llama (SentencePiece):** Llama-Tokenisierer verwenden die SentencePiece-Bibliothek, ein unüberwachter Text-Tokenisierer und Detokenisierer. SentencePiece behandelt den Eingabetext als eine Sequenz von Unicode-Zeichen und lernt ein Teilwort-Vokabular basierend auf einem Trainingskorpus. Es kann jede Sprache verarbeiten, die in Unicode kodiert werden kann, was es besonders geeignet für mehrsprachige Modelle macht. Llama-Tokenisierer werden von Modellen wie Meta's Llama und Alpaca verwendet.

- **SentencePiece (Unigram):** SentencePiece-Tokenisierer können auch einen anderen Algorithmus namens Unigram verwenden, der auf einer Teilwort-Regularisierungstechnik basiert. Die Unigram-Tokenisierung bestimmt das optimale Teilwort-Vokabular basierend auf einem Unigram-Sprachmodell, das einzelnen Teilworteinheiten Wahrscheinlichkeiten zuordnet. Dieser Ansatz kann im Vergleich zu BPE semantisch bedeutungsvollere Teilwörter erzeugen. SentencePiece mit Unigram wird von Modellen wie Google's T5 und BERT verwendet.
- **Google Gemini (Multimodale Tokenisierung):** Google Gemini verwendet ein Tokenisierungsschema, das für die Verarbeitung verschiedener Datentypen, einschließlich Text, Bilder, Audio, Videos und Code, entwickelt wurde. Diese multimodale Fähigkeit ermöglicht es Gemini, verschiedene Formen von Informationen zu verarbeiten und zu integrieren. Bemerkenswert ist, dass Google Gemini 1.5 Pro ein Kontextfenster hat, das Millionen von Token verarbeiten kann, deutlich mehr als frühere Modelle. Dieses umfangreiche Kontextfenster ermöglicht es dem Modell, einen größeren Kontext zu verarbeiten, was potenziell zu genaueren Antworten führt. Allerdings ist es wichtig zu beachten, dass Geminis Tokenisierungsschema viel näher an einem Token pro Zeichen liegt als bei anderen Modellen. Dies bedeutet, dass die tatsächlichen Kosten für die Nutzung von Gemini-Modellen deutlich höher sein können als erwartet, wenn man an Modelle wie GPT gewöhnt ist, da Googles Preisgestaltung auf Zeichen statt auf Token basiert.

Die Wahl des Tokenisierers beeinflusst verschiedene Aspekte eines LLM, darunter:

- **Vokabulargröße:** Der Tokenisierer bestimmt die Größe des Modellvokabulars, also die Menge der einzigartigen Token, die es erkennt. Ein größeres, feiner granuliertes Vokabular kann dem Modell helfen, eine breitere Palette von Wörtern und Phrasen zu verarbeiten und sogar multimodal zu werden (fähig, mehr als nur

Text zu verstehen und zu generieren), erhöht aber auch den Speicherbedarf und die Berechnungskomplexität des Modells.

- **Umgang mit seltenen und unbekannten Wörtern:** Tokenisierer, die Teilworteinheiten verwenden, wie BPE und SentencePiece, können seltene und unbekannte Wörter in häufigere Teilwörter zerlegen. Dies ermöglicht es dem Modell, fundierte Vermutungen über die Bedeutung von Wörtern anzustellen, die es noch nie gesehen hat, basierend auf den Teilwörtern, aus denen sie bestehen.
- **Mehrsprachige Unterstützung:** Tokenisierer wie SentencePiece, die jede Unicode-kodierbare Sprache verarbeiten können, sind gut geeignet für mehrsprachige Modelle, die Text in verschiedenen Sprachen verarbeiten müssen.

Bei der Auswahl eines LLM für eine bestimmte Anwendung ist es wichtig, den verwendeten Tokenisierer zu berücksichtigen und wie gut er mit den spezifischen Sprachverarbeitungsanforderungen der Aufgabe übereinstimmt. Der Tokenisierer kann einen erheblichen Einfluss auf die Fähigkeit des Modells haben, fachspezifische Terminologie, seltene Wörter und mehrsprachigen Text zu verarbeiten.

Kontextgröße: Wie viele Informationen kann ein Sprachmodell während der Inferenz nutzen?

Bei der Diskussion von Sprachmodellen bezieht sich die Kontextgröße auf die Textmenge, die ein Modell bei der Verarbeitung oder Generierung seiner Antworten berücksichtigen kann. Es ist im Wesentlichen ein Maß dafür, wie viel Information das Modell “erinnern” und für seine Ausgaben nutzen kann (ausgedrückt in Token). Die Kontextgröße eines Sprachmodells kann einen erheblichen Einfluss auf seine Fähigkeiten und die Arten von Aufgaben haben, die es effektiv ausführen kann.

Was ist Kontextgröße?

Technisch gesehen wird die Kontextgröße durch die Anzahl der Token (Wörter oder Wortteile) bestimmt, die ein Sprachmodell in einer einzelnen Eingabesequenz verarbeiten kann. Dies wird oft als “Aufmerksamkeitsspanne” oder “Kontextfenster” des Modells bezeichnet. Je größer die Kontextgröße, desto mehr Text kann das Modell gleichzeitig berücksichtigen, wenn es eine Antwort generiert oder eine Aufgabe ausführt.

Verschiedene Sprachmodelle haben unterschiedliche Kontextgrößen, die von einigen hundert Token bis zu Millionen von Token reichen. Zum Vergleich: Ein typischer Textabsatz enthält etwa 100-150 Token, während ein ganzes Buch Zehntausende oder Hunderttausende von Token enthalten kann.

Es gibt sogar Forschung zu effizienten Methoden, um Transformer-basierte Large Language Models (LLMs) auf **unendlich lange Eingaben** mit begrenztem Speicher und Rechenaufwand zu skalieren.

Warum ist die Kontextgröße wichtig?

Die Kontextgröße eines Sprachmodells hat einen erheblichen Einfluss auf seine Fähigkeit, zusammenhängenden und kontextuell relevanten Text zu verstehen und zu generieren. Hier sind einige wichtige Gründe, warum die Kontextgröße von Bedeutung ist:

1. **Verstehen längerer Inhalte:** Modelle mit größeren Kontextfenstern können längere Texte wie Artikel, Berichte oder sogar ganze Bücher besser verstehen und analysieren. Dies ist entscheidend für Aufgaben wie Dokumentenzusammenfassung, Fragenbeantwortung und Inhaltsanalyse.

2. **Aufrechterhaltung der Kohärenz:** Ein größeres Kontextfenster ermöglicht es dem Modell, Kohärenz und Konsistenz über längere Textabschnitte hinweg aufrechtzuerhalten. Dies ist wichtig für Aufgaben wie Geschichtengenerierung, Dialogsysteme und Content-Erstellung, bei denen die Beibehaltung einer konsistenten Erzählung oder Thematik wesentlich ist. Es ist auch absolut entscheidend bei der Verwendung von LLMs zur Generierung oder Transformation strukturierter Daten.
3. **Erfassung langreichweitiger Abhängigkeiten:** Einige sprachliche Aufgaben erfordern das Verständnis von Beziehungen zwischen Wörtern oder Phrasen, die im Text weit voneinander entfernt sind. Modelle mit größeren Kontextgrößen sind besser in der Lage, diese langreichweitigen Abhängigkeiten zu erfassen, was für Aufgaben wie Stimmungsanalyse, Übersetzung und Sprachverständnis wichtig sein kann.
4. **Umgang mit komplexen Anweisungen:** Bei Anwendungen, in denen Sprachmodelle komplexe, mehrstufige Anweisungen befolgen sollen, ermöglicht eine größere Kontextgröße dem Modell, den gesamten Satz von Anweisungen bei der Generierung einer Antwort zu berücksichtigen, anstatt nur die letzten paar Wörter.

Beispiele für Sprachmodelle mit unterschiedlichen Kontextgrößen

Hier sind einige Beispiele für Sprachmodelle mit unterschiedlichen Kontextgrößen:

- 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

Wie Sie sehen können, gibt es eine große Bandbreite an Kontextgrößen bei diesen Modellen, von etwa 4.000 Tokens beim OpenAI GPT-3.5 Turbo-Modell bis zu 200.000 Tokens beim Anthropic Claude v2-Modell. Einige Modelle, wie Googles PaLM 2 und OpenAIs GPT-4, bieten verschiedene Varianten mit größeren Kontextgrößen an (z.B. “32k”-Versionen), die noch längere Eingabesequenzen verarbeiten können. Und zum jetzigen Zeitpunkt (April 2024) prahlt Google Gemini Pro mit fast 3 Millionen Tokens!

Es ist erwähnenswert, dass die Kontextgröße je nach spezifischer Implementierung und Version eines bestimmten Modells variieren kann. Zum Beispiel hat das ursprüngliche OpenAI GPT-4-Modell eine Kontextgröße von 8.191 Tokens, während die späteren GPT-4-Varianten wie Turbo und 4o eine deutlich größere Kontextgröße von 128.000 Tokens aufweisen.

Sam Altman hat die aktuellen Kontextbeschränkungen mit den Kilobytes an Arbeitsspeicher verglichen, mit denen PC-Programmierer in den 80er Jahren umgehen mussten, und gesagt, dass wir in naher Zukunft in der Lage sein werden, “alle persönlichen Daten” in den Kontext eines großen Sprachmodells einzufügen.

Auswahl der richtigen Kontextgröße

Bei der Auswahl eines Sprachmodells für eine bestimmte Anwendung ist es wichtig, die Kontextgrößenanforderungen der jeweiligen Aufgabe zu berücksichtigen. Für Aufgaben, die kurze, isolierte Textstücke betreffen, wie Stimmungsanalyse oder einfache Fragenbeantwortung, kann eine kleinere Kontextgröße ausreichend sein. Für Aufgaben, die das Verstehen und Generieren längerer, komplexerer Texte erfordern, wird jedoch wahrscheinlich eine größere Kontextgröße notwendig sein.

Es ist zu beachten, dass größere Kontextgrößen oft mit erhöhten Rechenkosten und längeren Verarbeitungszeiten verbunden sind, da das Modell bei der Generierung einer

Antwort mehr Informationen berücksichtigen muss. Daher müssen Sie bei der Wahl eines Sprachmodells für Ihre Anwendung ein Gleichgewicht zwischen Kontextgröße und Leistung finden.

Warum wählt man nicht einfach das Modell mit der größten Kontextgröße und füttert es mit so vielen Informationen wie möglich? Nun, neben Leistungsfaktoren ist der andere Hauptaspekt die Kosten. Im März 2024 kostet ein *einzig* Prompt-Response-Zyklus mit Google Gemini Pro 1.5 bei vollem Kontext fast 8 USD. Wenn Sie einen Anwendungsfall haben, der diese Ausgaben rechtfertigt, nur zu! Aber für die meisten Anwendungen ist es um Größenordnungen zu teuer.

Nadeln im Heuhaufen finden

Das Konzept, eine Nadel im Heuhaufen zu finden, ist seit langem eine Metapher für die Herausforderungen beim Abruf in großen Datensätzen. Im Bereich der großen Sprachmodelle (LLMs) modifizieren wir diese Analogie ein wenig. Stellen Sie sich vor, wir suchen nicht nur nach einer einzelnen Information, die in einem umfangreichen Text verborgen ist (wie etwa in einer vollständigen Anthologie von Paul Graham Essays), sondern nach mehreren, über den Text verteilten Informationen. Dieses Szenario ähnelt eher der Suche nach mehreren Nadeln in einem weitläufigen Feld, nicht nur in einem einzelnen Heuhaufen. Und hier kommt's: Wir müssen diese Nadeln nicht nur finden, sondern sie auch zu einem zusammenhängenden Faden verweben.

Wenn LLMs damit beauftragt werden, mehrere in langen Kontexten eingebettete Informationen abzurufen und darüber zu schlussfolgern, stehen sie vor einer zweifachen Herausforderung. Erstens gibt es das offensichtliche Problem der Abrufgenauigkeit – sie nimmt naturgemäß ab, je mehr Informationen es gibt. Das ist zu erwarten; schließlich fordert die Verfolgung mehrerer Details in einem weitläufigen Text selbst die ausgefeiltesten Modelle heraus.

Zweitens, und vielleicht noch wichtiger, ist die Herausforderung, mit diesen Informationen zu argumentieren. Es ist eine Sache, Fakten herauszupicken; eine ganz andere ist es, sie zu einer kohärenten Erzählung oder Antwort zu synthetisieren. Hier kommt die wahre Prüfung. Die Leistung von LLMs bei Argumentationsaufgaben tendiert dazu, stärker abzunehmen als bei einfachen Abrufaufgaben. Diese Verschlechterung betrifft nicht nur das Volumen; es geht um das komplexe Zusammenspiel von Kontext, Relevanz und Schlussfolgerung.

Warum passiert das? Nun, betrachten wir die Dynamik von Gedächtnis und Aufmerksamkeit in der menschlichen Kognition, die sich bis zu einem gewissen Grad in LLMs widerspiegelt. Bei der Verarbeitung großer Informationsmengen können LLMs, ähnlich wie Menschen, frühere Details aus den Augen verlieren, während sie neue aufnehmen. Dies gilt besonders für Modelle, die nicht explizit darauf ausgelegt sind, frühere Textabschnitte automatisch zu priorisieren oder zu überprüfen.

Darüber hinaus ähnelt die Fähigkeit eines LLM, diese abgerufenen Fakten zu einer kohärenten Antwort zu verweben, dem Aufbau einer Erzählung. Dies erfordert nicht nur den Abruf von Informationen, sondern auch ein tiefes Verständnis und eine kontextuelle Einordnung, was für die aktuelle KI weiterhin eine große Herausforderung darstellt.

Was bedeutet das also für uns als Entwickler und Integratoren dieser Technologien? Wir müssen uns dieser Einschränkungen bewusst sein, wenn wir Systeme entwickeln, die sich auf LLMs für komplexe, längere Aufgaben verlassen. Das Verständnis, dass die Leistung unter bestimmten Bedingungen nachlassen könnte, hilft uns, realistische Erwartungen zu setzen und bessere Ausweichmechanismen oder ergänzende Strategien zu entwickeln.

Modalitäten: Jenseits des Textes

Während sich die Mehrheit der Sprachmodelle heute auf die Verarbeitung und Generierung von Text konzentriert, gibt es einen wachsenden Trend zu multimodalen Modellen, die verschiedene Arten von Daten wie Bilder, Audio und Video nativ ein-

und ausgeben können. Diese multimodalen Modelle eröffnen neue Möglichkeiten für KI-gestützte Anwendungen, die Inhalte über verschiedene Modalitäten hinweg verstehen und generieren können.

Was sind Modalitäten?

Im Kontext von Sprachmodellen beziehen sich Modalitäten auf die verschiedenen Arten von Daten, die ein Modell verarbeiten und generieren kann. Die häufigste Modalität ist Text, der geschriebene Sprache in verschiedenen Formen wie Bücher, Artikel, Websites und Social-Media-Beiträge umfasst. Es gibt jedoch mehrere andere Modalitäten, die zunehmend in Sprachmodelle integriert werden:

- **Bilder:** Visuelle Daten wie Fotografien, Illustrationen und Diagramme.
- **Audio:** Klanginformationen wie Sprache, Musik und Umgebungsgeräusche.
- **Video:** Bewegte visuelle Daten, oft begleitet von Audio, wie Videoclips und Filme.

Jede Modalität stellt eigene Herausforderungen und Chancen für Sprachmodelle dar. Beispielsweise muss das Modell bei Bildern visuelle Konzepte und Beziehungen verstehen, während es bei Audio Sprache und andere Geräusche verarbeiten und generieren muss.

Multimodale Sprachmodelle

Multimodale Sprachmodelle sind darauf ausgelegt, mehrere Modalitäten innerhalb eines einzelnen Modells zu verarbeiten. Diese Modelle verfügen typischerweise über spezialisierte Komponenten oder Schichten, die sowohl Eingaben verstehen als auch Ausgabedaten in verschiedenen Modalitäten generieren können. Einige bemerkenswerte Beispiele für multimodale Sprachmodelle sind:

- **OpenAI's GPT-4o:** GPT-4o ist ein großes Sprachmodell, das neben Text auch Sprachaudio nativ versteht und verarbeitet. Diese Fähigkeit ermöglicht es GPT-4o, Aufgaben wie die Transkription gesprochener Sprache, die Generierung von

Text aus Audioeingaben und die Bereitstellung von Antworten basierend auf gesprochenen Anfragen auszuführen.

- **OpenAI's GPT-4 mit visueller Eingabe:** GPT-4 ist ein großes Sprachmodell, das sowohl Text als auch Bilder verarbeiten kann. Wenn GPT-4 ein Bild als Eingabe erhält, kann es den Inhalt des Bildes analysieren und Text generieren, der die visuellen Informationen beschreibt oder darauf reagiert.
- **Google's Gemini:** Gemini ist ein multimodales Modell, das Text, Bilder und Video verarbeiten kann. Es verwendet eine einheitliche Architektur, die modalitätsübergreifendes Verstehen und Generieren ermöglicht und Aufgaben wie Bildbeschreibung, Videozusammenfassung und visuelle Fragenbeantwortung unterstützt.
- **DALL-E und Stable Diffusion:** Obwohl es sich nicht um Sprachmodelle im traditionellen Sinne handelt, demonstrieren diese Modelle die Leistungsfähigkeit multimodaler KI durch die Generierung von Bildern aus Textbeschreibungen. Sie zeigen das Potenzial von Modellen, die zwischen verschiedenen Modalitäten übersetzen können.

Vorteile und Anwendungen multimodaler Modelle

Multimodale Sprachmodelle bieten mehrere Vorteile und ermöglichen ein breites Spektrum an Anwendungen, darunter:

- **Verbessertes Verständnis:** Durch die Verarbeitung von Informationen aus mehreren Modalitäten können diese Modelle ein umfassenderes Verständnis der Welt entwickeln, ähnlich wie Menschen aus verschiedenen Sinneseindrücken lernen.
- **Modalitätsübergreifende Generierung:** Multimodale Modelle können Inhalte in einer Modalität basierend auf Eingaben aus einer anderen generieren, wie zum Beispiel die Erstellung eines Bildes aus einer Textbeschreibung oder die Generierung einer Videozusammenfassung aus einem geschriebenen Artikel.

- **Barrierefreiheit:** Multimodale Modelle können Informationen zugänglicher machen, indem sie zwischen Modalitäten übersetzen, wie etwa durch die Generierung von Textbeschreibungen von Bildern für sehbehinderte Nutzer oder die Erstellung von Audio-Versionen schriftlicher Inhalte.
- **Kreative Anwendungen:** Multimodale Modelle können für kreative Aufgaben wie die Generierung von Kunst, Musik oder Videos basierend auf textuellen Eingabeaufforderungen verwendet werden, was neue Möglichkeiten für Künstler und Content-Ersteller eröffnet.

Mit der fortschreitenden Entwicklung multimodaler Sprachmodelle werden sie voraussichtlich eine zunehmend wichtige Rolle bei der Entwicklung KI-gestützter Anwendungen spielen, die Inhalte über mehrere Modalitäten hinweg verstehen und generieren können. Dies wird natürlichere und intuitivere Interaktionen zwischen Menschen und KI-Systemen ermöglichen sowie neue Möglichkeiten für kreativen Ausdruck und Wissensvermittlung erschließen.

Anbieter-Ökosysteme

Wenn es darum geht, große Sprachmodelle (LLMs) in Anwendungen zu integrieren, steht eine wachsende Auswahl an Optionen zur Verfügung. Jeder große LLM-Anbieter, wie OpenAI, Anthropic, Google und Cohere, bietet sein eigenes Ökosystem von Modellen, APIs und Werkzeugen an. Die Wahl des richtigen Anbieters erfordert die Berücksichtigung verschiedener Faktoren, einschließlich Preisgestaltung, Leistung, Inhaltsfilterung, Datenschutz und Anpassungsoptionen.

OpenAI

OpenAI ist einer der bekanntesten Anbieter von LLMs, wobei seine GPT-Serie (GPT-3, GPT-4) in verschiedenen Anwendungen weit verbreitet ist. OpenAI bietet eine benutzerfreundliche API, die es ermöglicht, ihre Modelle einfach in Anwendungen zu

integrieren. Sie bieten eine Reihe von Modellen mit unterschiedlichen Fähigkeiten und Preispunkten an, vom Einstiegsmodell Ada bis zum leistungsstarken Davinci-Modell.

Das Ökosystem von OpenAI umfasst auch Werkzeuge wie den OpenAI Playground, der es ermöglicht, mit Prompts zu experimentieren und Modelle für spezifische Anwendungsfälle feinabzustimmen. Sie bieten Inhaltsfilterungsoptionen, um die Generierung unangemessener oder schädlicher Inhalte zu verhindern.

Bei der direkten Verwendung von OpenAI-Modellen verlasse ich mich auf Alex Rudalls [ruby-openai](#) Bibliothek.

Anthropic

Anthropic ist ein weiterer wichtiger Akteur im LLM-Bereich, dessen Claude-Modelle aufgrund ihrer starken Leistung und ethischen Überlegungen an Popularität gewinnen. Anthropic konzentriert sich auf die Entwicklung sicherer und verantwortungsvoller KI-Systeme, mit starkem Fokus auf Inhaltsfilterung und Vermeidung schädlicher Ausgaben.

Das Ökosystem von Anthropic umfasst die Claude-API, die es ermöglicht, das Modell in ihre Anwendungen zu integrieren, sowie Werkzeuge für Prompt-Engineering und Feinabstimmung. Sie bieten auch das Claude Instant-Modell an, das Websuchfunktionen für aktuellere und faktisch genauere Antworten integriert.

Bei der direkten Verwendung von Anthropic-Modellen verlasse ich mich auf Alex Rudalls [anthropic](#) Bibliothek.

Google

Google hat mehrere leistungsstarke LLMs entwickelt, darunter Gemini, BERT, T5 und PaLM. Diese Modelle sind für ihre starke Leistung bei einer Vielzahl von Aufgaben zur Verarbeitung natürlicher Sprache bekannt. Das Google-Ökosystem umfasst die TensorFlow- und Keras-Bibliotheken, die Werkzeuge und Frameworks für den Aufbau und das Training von maschinellen Lernmodellen bereitstellen.

Google bietet auch eine Cloud-KI-Plattform an, die es ermöglicht, ihre Modelle einfach in der Cloud zu implementieren und zu skalieren. Sie stellen eine Reihe vortrainierter Modelle und APIs für Aufgaben wie Stimmungsanalyse, Entitätserkennung und Übersetzung zur Verfügung.

Meta

Meta, früher als Facebook bekannt, investiert stark in die Entwicklung großer Sprachmodelle, was durch die Veröffentlichung von Modellen wie LLaMA und OPT deutlich wird. Diese Modelle zeichnen sich durch ihre starke Leistung bei verschiedenen Sprachaufgaben aus und werden größtenteils über Open-Source-Kanäle verfügbar gemacht, was Metas Engagement für Forschung und gemeinschaftliche Zusammenarbeit unterstreicht.

Das Ökosystem von Meta basiert hauptsächlich auf PyTorch, einer Open-Source-Bibliothek für maschinelles Lernen, die für ihre dynamischen Berechnungsmöglichkeiten und Flexibilität geschätzt wird und innovative KI-Forschung und -Entwicklung ermöglicht.

Neben ihren technischen Angeboten legt Meta großen Wert auf die ethische Entwicklung von KI. Sie implementieren robuste Inhaltsfilterung und konzentrieren sich darauf, Verzerrungen zu reduzieren, was mit ihren übergeordneten Zielen der Sicherheit und Verantwortung in KI-Anwendungen übereinstimmt.

Cohere

Cohere ist ein neuerer Teilnehmer im LLM-Bereich, der sich darauf konzentriert, LLMs zugänglicher und einfacher zu nutzen als die Konkurrenz. Ihr Ökosystem umfasst die Cohere API, die Zugang zu einer Reihe vortrainierter Modelle für Aufgaben wie Texterstellung, Klassifizierung und Zusammenfassung bietet.

Cohere bietet auch Werkzeuge für Prompt-Engineering, Feinabstimmung und

Inhaltsfilterung an. Sie legen besonderen Wert auf Datenschutz und Sicherheit, mit Funktionen wie verschlüsselter Datenspeicherung und Zugriffskontrollen.

Ollama

Ollama ist eine selbst gehostete Plattform, die es Benutzern ermöglicht, verschiedene große Sprachmodelle (LLMs) lokal auf ihren Maschinen zu verwalten und einzusetzen, wodurch sie vollständige Kontrolle über ihre KI-Modelle haben, ohne auf externe Cloud-Dienste angewiesen zu sein. Diese Einrichtung ist ideal für diejenigen, die Datenschutz priorisieren und ihre KI-Operationen intern handhaben möchten.

Die Plattform unterstützt eine Reihe von Modellen, einschließlich Versionen von Llama, Phi, Gemma und Mistral, die sich in Größe und Rechenanforderungen unterscheiden. Ollama macht es einfach, diese Modelle direkt von der Befehlszeile mit einfachen Befehlen wie `ollama run <model_name>` herunterzuladen und auszuführen, und ist für verschiedene Betriebssysteme wie macOS, Linux und Windows konzipiert.

Für Entwickler, die Open-Source-Modelle in ihre Anwendungen integrieren möchten, ohne eine Remote-API zu verwenden, bietet Ollama eine CLI zur Verwaltung von Modell-Lebenszyklen, ähnlich wie Container-Management-Tools. Es unterstützt auch benutzerdefinierte Konfigurationen und Prompts, was ein hohes Maß an Anpassung ermöglicht, um die Modelle auf spezifische Bedürfnisse oder Anwendungsfälle zuzuschneiden.

Ollama ist besonders für technisch versierte Benutzer und Entwickler geeignet, da es eine Befehlszeilenschnittstelle und Flexibilität bei der Verwaltung und Bereitstellung von KI-Modellen bietet. Dies macht es zu einem leistungsfähigen Werkzeug für Unternehmen und Einzelpersonen, die robuste KI-Fähigkeiten benötigen, ohne Kompromisse bei Sicherheit und Kontrolle einzugehen.

Multi-Modell-Plattformen

Zusätzlich gibt es Anbieter, die eine große Vielfalt an Open-Source-Modellen hosten, wie Together.ai und Groq.. Diese Plattformen bieten Flexibilität und Anpassungsmöglichkeiten und erlauben es Ihnen, Open-Source-Modelle auszuführen und in manchen Fällen sogar nach Ihren spezifischen Bedürfnissen feinabzustimmen. Together.ai bietet beispielsweise Zugang zu einer Reihe von Open-Source-LLMs und ermöglicht es Benutzern, mit verschiedenen Modellen und Konfigurationen zu experimentieren. Groq konzentriert sich darauf, ultrahohe Verarbeitungsgeschwindigkeiten zu liefern, die zum Zeitpunkt der Entstehung dieses Buches fast magisch erscheinen

Auswahl eines LLM-Anbieters

Bei der Auswahl eines LLM-Anbieters sollten Sie folgende Faktoren berücksichtigen:

- **Preisgestaltung:** Verschiedene Anbieter bieten unterschiedliche Preismodelle an, von nutzungsbasierter Abrechnung bis hin zu Abonnement-basierten Plänen. Es ist wichtig, die erwartete Nutzung und das Budget bei der Auswahl eines Anbieters zu berücksichtigen.
- **Leistung:** Die Leistung von LLMs kann zwischen Anbietern erheblich variieren, daher ist es wichtig, Modelle für spezifische Anwendungsfälle zu testen und zu bewerten, bevor eine Entscheidung getroffen wird.
- **Inhaltsfilterung:** Je nach Anwendung kann Inhaltsfilterung ein kritischer Aspekt sein. Einige Anbieter bieten robustere Inhaltsfilterungsoptionen als andere.
- **Datenschutz:** Wenn die Anwendung sensible Benutzerdaten verarbeitet, ist es wichtig, einen Anbieter mit strengen Datenschutz- und Sicherheitspraktiken zu wählen.
- **Anpassung:** Einige Anbieter bieten mehr Flexibilität bei der Feinabstimmung und Anpassung von Modellen für spezifische Anwendungsfälle.

Letztendlich hängt die Wahl des LLM-Anbieters von den spezifischen Anforderungen und Einschränkungen der Anwendung ab. Durch sorgfältige Bewertung der Optionen und Berücksichtigung von Faktoren wie Preisgestaltung, Leistung und Datenschutz können Sie den Anbieter auswählen, der Ihre Bedürfnisse am besten erfüllt.

Es ist auch erwähnenswert, dass sich die LLM-Landschaft ständig weiterentwickelt, wobei regelmäßig neue Anbieter und Modelle auftauchen. Sie sollten sich über die neuesten Entwicklungen auf dem Laufenden halten und offen für die Erkundung neuer Optionen sein, wenn diese verfügbar werden.

OpenRouter

In diesem Buch werde ich ausschließlich [OpenRouter](#) als meinen API-Anbieter der Wahl verwenden. Der Grund ist einfach: Es ist eine zentrale Anlaufstelle für alle populärsten kommerziellen und Open-Source-Modelle. Wenn Sie darauf brennen, mit etwas KI-Programmierung zu beginnen, ist einer der besten Ausgangspunkte meine eigene [OpenRouter Ruby Library](#).

Überlegungen zur Leistung

Bei der Integration von Sprachmodellen in Anwendungen ist die Leistung ein entscheidender Faktor. Die Leistung eines Sprachmodells kann anhand seiner *Latenz* (die Zeit, die für die Generierung einer Antwort benötigt wird) und seines *Durchsatzes* (die Anzahl der Anfragen, die pro Zeiteinheit verarbeitet werden können) gemessen werden.

Zeit bis zum ersten Token (TTFT) ist eine weitere wichtige Leistungskennzahl, die besonders für Chatbots und Anwendungen relevant ist, die interaktive Echtzeitantworten erfordern. TTFT misst die Latenz vom Moment des Eingangs einer Benutzeranfrage bis zum Moment, in dem das erste Wort (oder Token) der Antwort generiert wird. Diese Kennzahl ist entscheidend für die Aufrechterhaltung

einer nahtlosen und fesselnden Benutzererfahrung, da verzögerte Antworten zu Frustration und nachlassendem Engagement der Benutzer führen können.

Diese Leistungskennzahlen können einen erheblichen Einfluss auf die Benutzererfahrung und die Skalierbarkeit der Anwendung haben.

Mehrere Faktoren können die Leistung eines Sprachmodells beeinflussen:

Parameteranzahl: Größere Modelle mit mehr Parametern benötigen im Allgemeinen mehr Rechenressourcen und können im Vergleich zu kleineren Modellen eine höhere Latenz und einen geringeren Durchsatz aufweisen.

Hardware: Die Leistung eines Sprachmodells kann je nach der verwendeten Hardware erheblich variieren. Cloud-Anbieter bieten GPU- und TPU-Instanzen an, die für Machine-Learning-Workloads optimiert sind und die Modellinferenz erheblich beschleunigen können.



Einer der Vorteile von OpenRouter ist, dass man bei vielen der angebotenen Modelle die Wahl zwischen verschiedenen Cloud-Anbietern mit unterschiedlichen Leistungsprofilen und Kosten hat.

Quantisierung: Quantisierungstechniken können verwendet werden, um den Speicherbedarf und die Rechenanforderungen eines Modells zu reduzieren, indem Gewichte und Aktivierungen mit Datentypen niedrigerer Präzision dargestellt werden. Dies kann die Leistung verbessern, ohne die Qualität wesentlich zu beeinträchtigen. Als Anwendungsentwickler werden Sie sich wahrscheinlich nicht mit dem Training eigener Modelle auf verschiedenen Quantisierungsebenen beschäftigen, aber es ist gut, zumindest mit der Terminologie vertraut zu sein.

Batch-Verarbeitung: Die gleichzeitige Verarbeitung mehrerer Anfragen in Batches kann den Durchsatz verbessern, indem der Overhead für das Laden von Modellen und die Datenübertragung amortisiert wird.

Zwischenspeicherung: Das Zwischenspeichern der Ergebnisse häufig verwendeter Prompts oder Eingabesequenzen kann die Anzahl der Inferenzanfragen reduzieren und die Gesamtleistung verbessern.

Bei der Auswahl eines Sprachmodells für eine Produktionsanwendung ist es wichtig, dessen Leistung anhand repräsentativer Workloads und Hardwarekonfigurationen zu bewerten. Dies kann helfen, potenzielle Engpässe zu identifizieren und sicherzustellen, dass das Modell die erforderlichen Leistungsziele erreichen kann.

Es lohnt sich auch, die Kompromisse zwischen Modellleistung und anderen Faktoren wie Kosten, Flexibilität und Integrationsaufwand zu berücksichtigen. Beispielsweise kann die Verwendung eines kleineren, kostengünstigeren Modells mit geringerer Latenz für Anwendungen, die Echtzeitantworten erfordern, vorzuziehen sein, während ein größeres, leistungsfähigeres Modell besser für Batch-Verarbeitung oder komplexe Denkaufgaben geeignet sein kann.

Experimentieren mit verschiedenen LLM-Modellen

Die Wahl eines LLM ist selten eine endgültige Entscheidung. Da regelmäßig neue und verbesserte Modelle veröffentlicht werden, ist es gut, Anwendungen modular zu entwickeln, sodass verschiedene Sprachmodelle im Laufe der Zeit ausgetauscht werden können. Prompts und Datensätze können oft mit minimalen Änderungen über verschiedene Modelle hinweg wiederverwendet werden. Dies ermöglicht es, die neuesten Fortschritte in der Sprachmodellierung zu nutzen, ohne die Anwendungen komplett neu gestalten zu müssen.



Die Möglichkeit, einfach zwischen einer großen Auswahl an Modellen zu wechseln, ist ein weiterer Grund, warum ich OpenRouter so schätze.

Bei der Aktualisierung auf ein neues Sprachmodell ist es wichtig, dessen Leistung und

Ausgabequalität gründlich zu testen und zu validieren, um sicherzustellen, dass es die Anforderungen der Anwendung erfüllt. Dies kann das Nachtraining oder Fine-Tuning des Modells mit domänenspezifischen Daten sowie die Aktualisierung nachgelagerter Komponenten, die von den Ausgaben des Modells abhängen, umfassen.

Durch die Entwicklung von Anwendungen unter Berücksichtigung von Leistung und Modularität können Sie skalierbare, effiziente und zukunftssichere Systeme erstellen, die sich an die sich schnell entwickelnde Landschaft der Sprachmodellierungstechnologie anpassen können.

Zusammengesetzte KI-Systeme

Bevor wir unsere Einführung abschließen, ist es erwähnenswert, dass vor 2023 und dem durch ChatGPT ausgelösten Interesse an generativer KI traditionelle KI-Ansätze meist auf der Integration einzelner, geschlossener Modelle basierten. Im Gegensatz dazu nutzen *Zusammengesetzte KI-Systeme* komplexe Pipelines miteinander verbundener Komponenten, die zusammenarbeiten, um intelligentes Verhalten zu erreichen.

Im Kern bestehen zusammengesetzte KI-Systeme aus mehreren Modulen, von denen jedes für die Ausführung spezifischer Aufgaben oder Funktionen konzipiert ist. Diese Module können Generatoren, Abrufsysteme, Rangiersysteme, Klassifikatoren und verschiedene andere spezialisierte Komponenten umfassen. Indem das Gesamtsystem in kleinere, fokussierte Einheiten aufgeteilt wird, können Entwickler flexiblere, skalierbarere und wartbarere KI-Architekturen erstellen.

Einer der wichtigsten Vorteile von zusammengesetzten KI-Systemen ist ihre Fähigkeit, die Stärken verschiedener KI-Techniken und -Modelle zu kombinieren. Ein System könnte zum Beispiel ein großes Sprachmodell (LLM) für das Verstehen und Generieren natürlicher Sprache verwenden und gleichzeitig ein separates Modell für Informationsabruf oder regelbasierte Entscheidungsfindung einsetzen. Dieser modulare Ansatz ermöglicht es Ihnen, die besten Werkzeuge und Techniken für jede spezifische Aufgabe auszuwählen, anstatt sich auf eine Universallösung zu verlassen.

Allerdings bringt der Aufbau von zusammengesetzten KI-Systemen auch besondere Herausforderungen mit sich. Insbesondere erfordert die Sicherstellung der allgemeinen Kohärenz und Konsistenz des Systemverhaltens robuste Test-, Überwachungs- und Steuerungsmechanismen.



Das Aufkommen leistungsfähiger LLMs wie GPT-4 ermöglicht es uns, einfacher als je zuvor mit zusammengesetzten KI-Systemen zu experimentieren, da diese fortgeschrittenen Modelle in der Lage sind, mehrere Rollen innerhalb eines zusammengesetzten Systems zu übernehmen, wie Klassifizierung, Ranking und Generierung, zusätzlich zu ihren Fähigkeiten zum Verstehen natürlicher Sprache. Diese Vielseitigkeit ermöglicht es Entwicklern, schnell Prototypen zu erstellen und an zusammengesetzten KI-Architekturen zu iterieren, was neue Möglichkeiten für die Entwicklung intelligenter Anwendungen eröffnet.

Bereitstellungsmuster für zusammengesetzte KI-Systeme

Zusammengesetzte KI-Systeme können mit verschiedenen Mustern bereitgestellt werden, die jeweils für spezifische Anforderungen und Anwendungsfälle konzipiert sind. Betrachten wir vier häufige Bereitstellungsmuster: Frage und Antwort, Multi-Agent/Agentische Problemlöser, Konversations-KI und CoPilots.

Frage und Antwort

Frage-und-Antwort-Systeme (Q&A) konzentrieren sich darauf, Informationsabruf bereitzustellen, der durch die Verständnisfähigkeiten von KI-Modellen erweitert wird, um mehr als nur eine Suchmaschine zu sein. Durch die Kombination leistungsfähiger Sprachmodelle mit externen Wissensquellen unter Verwendung von [Retrieval-Augmented Generation \(RAG\)](#) vermeiden Q&A-Systeme Halluzinationen und liefern genaue und kontextuell relevante Antworten auf Benutzeranfragen.

Die Schlüsselkomponenten eines LLM-basierten Q&A-Systems umfassen:

- **Anfrageverständnis und -umformulierung:** Analyse von Benutzeranfragen und deren Umformulierung, um besser zu den zugrunde liegenden Wissensquellen zu passen.
- **Wissensabruf:** Abrufen relevanter Informationen aus strukturierten oder unstrukturierten Datenquellen basierend auf der umformulierten Anfrage.
- **Antwortgenerierung:** Generierung kohärenter und informativer Antworten durch Integration des abgerufenen Wissens mit den generativen Fähigkeiten des Sprachmodells.

RAG-Subsysteme sind besonders wichtig in Q&A-Bereichen, in denen die Bereitstellung genauer und aktueller Informationen entscheidend ist, wie im Kundenservice, Wissensmanagement oder bei Bildungsanwendungen.

Multi-Agent/Agentische Problemlöser

Multi-Agent- oder auch *agentische* Systeme bestehen aus mehreren autonomen Agenten, die zusammenarbeiten, um komplexe Probleme zu lösen. Jeder Agent hat eine spezifische Rolle, einen Satz von Fähigkeiten und Zugang zu relevanten Werkzeugen oder Informationsquellen. Durch Zusammenarbeit und Informationsaustausch können diese Agenten Aufgaben bewältigen, die für einen einzelnen Agenten schwierig oder unmöglich zu handhaben wären.

Die Schlüsselprinzipien von Multi-Agent-Problemlösern umfassen:

- **Spezialisierung:** Jeder Agent konzentriert sich auf einen spezifischen Aspekt des Problems und nutzt dabei seine einzigartigen Fähigkeiten und sein Wissen.
- **Zusammenarbeit:** Agenten kommunizieren und koordinieren ihre Aktionen, um ein gemeinsames Ziel zu erreichen, oft durch Nachrichtenaustausch oder gemeinsamen Speicher.

- **Anpassungsfähigkeit:** Das System kann sich an veränderte Bedingungen oder Anforderungen anpassen, indem es die Rollen und Verhaltensweisen einzelner Agenten anpasst.

Multi-Agent-Systeme eignen sich gut für Anwendungen, die verteilte Problemlösung erfordern, wie Lieferkettenoptimierung, Verkehrsmanagement oder Notfalleinsatzplanung.

Konversations-KI

Konversations-KI-Systeme ermöglichen natürlichsprachliche Interaktionen zwischen Benutzern und intelligenten Agenten. Diese Systeme kombinieren Sprachverständnis, Dialogverwaltung und Sprachgenerierung, um ansprechende und personalisierte Konversationserlebnisse zu bieten.

Die Hauptkomponenten eines Konversations-KI-Systems umfassen:

- **Intentionserkennung:** Identifizierung der Benutzerabsicht basierend auf ihrer Eingabe, wie das Stellen einer Frage, das Äußern einer Anfrage oder das Ausdrücken eines Gefühls.
- **Entitätsextraktion:** Extraktion relevanter Entitäten oder Parameter aus der Benutzereingabe, wie Daten, Orte oder Produktnamen.
- **Dialogverwaltung:** Aufrechterhaltung des Gesprächszustands, Bestimmung der angemessenen Antwort basierend auf der Benutzerabsicht und dem Kontext sowie Handhabung von Mehrfachinteraktionen.
- **Antwortgenerierung:** Generierung menschenähnlicher Antworten unter Verwendung von Sprachmodellen, Vorlagen oder abrufbasierten Methoden.

Konversations-KI-Systeme werden häufig in Kundenservice-Chatbots, virtuellen Assistenten und sprachgesteuerten Schnittstellen eingesetzt. Wie bereits erwähnt, stammen die meisten Ansätze, Muster und Codebeispiele in diesem Buch direkt aus meiner Arbeit an einem großen Konversations-KI-System namens [Olympia](#).

CoPilots

CoPilots sind KI-gestützte Assistenten, die zusammen mit menschlichen Benutzern arbeiten, um deren Produktivität und Entscheidungsfindung zu verbessern. Diese Systeme nutzen eine Kombination aus Natursprachverarbeitung, maschinellem Lernen und domänenspezifischem Wissen, um intelligente Empfehlungen zu geben, Aufgaben zu automatisieren und kontextbezogene Unterstützung zu bieten.

Zu den wichtigsten Merkmalen von CoPilots gehören:

- **Personalisierung:** Anpassung an individuelle Benutzerpräferenzen, Arbeitsabläufe und Kommunikationsstile.
- **Proaktive Unterstützung:** Vorwegnahme von Benutzerbedürfnissen und Angebot relevanter Vorschläge oder Aktionen ohne ausdrückliche Aufforderung.
- **Kontinuierliches Lernen:** Verbesserung der Leistung im Laufe der Zeit durch das Lernen aus Benutzer-Feedback, Interaktionen und Daten.

CoPilots werden zunehmend in verschiedenen Bereichen eingesetzt, wie beispielsweise in der Softwareentwicklung (z.B. Code-Vervollständigung und Fehlererkennung), im kreativen Schreiben (z.B. Inhaltsvorschläge und Bearbeitung) und in der Datenanalyse (z.B. Erkenntnisse und Visualisierungsempfehlungen)

Diese Einsatzmuster zeigen die Vielseitigkeit und das Potenzial von zusammengesetzten KI-Systemen. Durch das Verständnis der Eigenschaften und Anwendungsfälle jedes Musters können Sie fundierte Entscheidungen bei der Gestaltung und Implementierung intelligenter Anwendungen treffen. Obwohl sich dieses Buch nicht speziell mit der Implementierung von zusammengesetzten KI-Systemen befasst, gelten viele, wenn nicht alle der gleichen Ansätze und Muster für die Integration diskreter KI-Komponenten in die ansonsten traditionelle Anwendungsentwicklung.

Rollen in zusammengesetzten KI-Systemen

Zusammengesetzte KI-Systeme basieren auf einer Grundlage miteinander verbundener Module, die jeweils für die Ausführung einer bestimmten Rolle konzipiert sind. Diese Module arbeiten zusammen, um intelligentes Verhalten zu erzeugen und komplexe Probleme zu lösen. Es ist nützlich, diese Rollen zu kennen, wenn man darüber nachdenkt, wo man möglicherweise Teile seiner Anwendung mit diskreten KI-Komponenten implementieren oder ersetzen könnte.

Generator

Generatoren sind dafür verantwortlich, neue Daten oder Inhalte basierend auf gelernten Mustern oder Eingabeaufforderungen zu erzeugen. Die KI-Welt verfügt über viele verschiedene Arten von Generatoren, aber im Kontext der Sprachmodelle, die in diesem Buch vorgestellt werden, können Generatoren menschenähnlichen Text erstellen, unvollständige Sätze vervollständigen oder Antworten auf Benutzeranfragen generieren. Sie spielen eine entscheidende Rolle bei Aufgaben wie der Inhaltserstellung, Dialoggenerierung und Datenerweiterung.

Retriever

Retriever werden verwendet, um relevante Informationen aus großen Datensätzen oder Wissensdatenbanken zu suchen und zu extrahieren. Sie verwenden Techniken wie semantische Suche, Keyword-Matching oder Vektor-Ähnlichkeit, um die relevantesten Datenpunkte basierend auf einer gegebenen Anfrage oder einem Kontext zu finden. Retriever sind essentiell für Aufgaben, die einen schnellen Zugriff auf spezifische Informationen erfordern, wie beispielsweise Frage-Antwort-Systeme, Faktenüberprüfung oder Inhaltsempfehlungen.

Ranker

Ranker sind für die Ordnung oder Priorisierung einer Menge von Elementen basierend auf bestimmten Kriterien oder Relevanzwerten verantwortlich. Sie weisen jedem Element Gewichtungen oder Punktzahlen zu und sortieren sie entsprechend. Ranker werden häufig in Suchmaschinen, Empfehlungssystemen oder in jeder Anwendung eingesetzt, bei der es wichtig ist, den Benutzern die relevantesten Ergebnisse zu präsentieren.

Classifier

Classifier werden verwendet, um Datenpunkte basierend auf vordefinierten Klassen oder Kategorien zu kategorisieren oder zu kennzeichnen. Sie lernen aus gekennzeichneten Trainingsdaten und sagen dann die Klasse neuer, ungesehener Instanzen vorher. Classifier sind fundamental für Aufgaben wie Stimmungsanalyse, Spam-Erkennung oder Bilderkennung, bei denen das Ziel darin besteht, jeder Eingabe eine spezifische Kategorie zuzuweisen.

Werkzeuge & Agenten

Zusätzlich zu diesen Kernrollen integrieren zusammengesetzte KI-Systeme häufig Werkzeuge und Agenten, um ihre Funktionalität und Anpassungsfähigkeit zu verbessern:

- **Werkzeuge:** Werkzeuge sind diskrete Softwarekomponenten oder APIs, die spezifische Aktionen oder Berechnungen ausführen. Sie können von anderen Modulen wie Generatoren oder Retrievern aufgerufen werden, um Teilaufgaben zu erfüllen oder zusätzliche Informationen zu sammeln. Beispiele für Werkzeuge sind Websuchmaschinen, Taschenrechner oder Datenvisualisierungsbibliotheken.
- **Agenten:** Agenten sind autonome Einheiten, die ihre Umgebung wahrnehmen, Entscheidungen treffen und Aktionen ausführen können, um bestimmte Ziele

zu erreichen. Sie stützen sich häufig auf eine Kombination verschiedener KI-Techniken wie Planung, Reasoning und Lernen, um in dynamischen oder unsicheren Bedingungen effektiv zu arbeiten. Agenten können verwendet werden, um komplexe Verhaltensweisen zu modellieren oder die Aktionen mehrerer Module innerhalb eines zusammengesetzten KI-Systems zu koordinieren.

In einem reinen zusammengesetzten KI-System wird die Interaktion zwischen diesen Komponenten durch klar definierte Schnittstellen und Kommunikationsprotokolle orchestriert. Daten fließen zwischen den Modulen, wobei die Ausgabe einer Komponente als Eingabe für eine andere dient. Diese modulare Architektur ermöglicht Flexibilität, Skalierbarkeit und Wartbarkeit, da einzelne Komponenten aktualisiert, ersetzt oder erweitert werden können, ohne das gesamte System zu beeinflussen.

Durch die Nutzung der Leistungsfähigkeit dieser Komponenten und ihrer Interaktionen können zusammengesetzte KI-Systeme komplexe, reale Probleme bewältigen, die eine Kombination verschiedener KI-Fähigkeiten erfordern. Während wir die Ansätze und Muster für die Integration von KI in die Anwendungsentwicklung erkunden, sollten Sie bedenken, dass die gleichen Prinzipien und Techniken, die in zusammengesetzten KI-Systemen verwendet werden, auch für die Erstellung intelligenter, adaptiver und benutzerzentrierter Anwendungen genutzt werden können.

In den folgenden Kapiteln von Teil 1 werden wir tiefer in die grundlegenden Ansätze und Techniken für die Integration von KI-Komponenten in Ihren Anwendungsentwicklungsprozess eintauchen. Von Prompt-Engineering und abrufgestützter Generierung bis hin zu selbstheilenden Daten und intelligenter Workflow-Orchestrierung werden wir eine breite Palette von Mustern und Best Practices behandeln, die Ihnen beim Aufbau modernster KI-gestützter Anwendungen helfen.

Teil 1: Grundlegende Ansätze & Techniken

Dieser Teil des Buches stellt verschiedene Möglichkeiten vor, wie Sie KI in Ihre Anwendungen integrieren können. Die Kapitel behandeln eine Reihe verwandter Ansätze und Techniken, von übergeordneten Konzepten wie [Den Pfad eingrenzen](#) und [Retrieval Augmented Generation](#) bis hin zu Ideen für die Programmierung einer eigenen Abstraktionsschicht über LLM Chat Completion APIs.

Ziel dieses Buchteils ist es, Ihnen zu vermitteln, welche Arten von Verhalten Sie mit KI implementieren können, bevor Sie sich zu tief in die spezifischen Implementierungsmuster vertiefen, die den Schwerpunkt von [Teil 2](#) bilden.

Die Ansätze in Teil 1 basieren auf Ideen, die ich in meinem Code verwendet habe, klassischen Mustern der Unternehmensanwendungsarchitektur und -integration sowie Metaphern, die ich beim Erklären der KI-Fähigkeiten gegenüber anderen Personen, einschließlich nicht-technischer Geschäftspartner, verwendet habe.

Den Pfad eingrenzen



“Den Pfad eingrenzen” bezieht sich darauf, die KI auf die aktuelle Aufgabe zu fokussieren. Ich verwende es als Mantra, wann immer ich frustriert bin, weil die KI sich “dumm” oder unerwartet verhält. Das Mantra erinnert mich daran, dass der Fehler wahrscheinlich bei mir liegt und dass ich den Pfad vermutlich noch weiter eingrenzen sollte.

Die Notwendigkeit, den Pfad einzugrenzen, ergibt sich aus der gewaltigen Menge an Wissen, die in großen Sprachmodellen enthalten ist, insbesondere in Weltklasse-Modellen wie denen von OpenAI und Anthropic, die buchstäblich Billionen von

Parametern haben.

Der Zugang zu einem solch breiten Wissensspektrum ist zweifellos mächtig und erzeugt emergentes Verhalten wie Theory of Mind und die Fähigkeit, auf menschenähnliche Weise zu denken. Allerdings stellt diese erschütternde Informationsmenge auch Herausforderungen dar, wenn es darum geht, präzise und akkurate Antworten auf spezifische Prompts zu generieren, besonders wenn diese Prompts deterministisches Verhalten aufweisen sollen, das in die "normale" Softwareentwicklung und Algorithmen integriert werden kann.

Mehrere Faktoren führen zu diesen Herausforderungen.

Informationsüberflutung: Große Sprachmodelle werden mit riesigen Datenmengen trainiert, die verschiedene Bereiche, Quellen und Zeiträume umfassen. Dieses umfangreiche Wissen ermöglicht es ihnen, sich mit verschiedenen Themen auseinanderzusetzen und Antworten basierend auf einem breiten Weltverständnis zu generieren. Wenn das Modell jedoch mit einem spezifischen Prompt konfrontiert wird, könnte es Schwierigkeiten haben, irrelevante, widersprüchliche oder veraltete/obsolete Informationen herauszufiltern, was zu Antworten führt, denen es an Fokus oder Genauigkeit mangelt. Je nachdem, was Sie zu erreichen versuchen, kann die schiere Menge an *widersprüchlichen* Informationen, die dem Modell zur Verfügung stehen, leicht seine Fähigkeit überfordern, die gewünschte Antwort oder das gewünschte Verhalten zu liefern.

Kontextuelle Mehrdeutigkeit: Angesichts des riesigen *latenten Raums* an Wissen können große Sprachmodelle auf Mehrdeutigkeiten stoßen, wenn sie versuchen, den *Kontext* Ihres Prompts zu verstehen. Ohne angemessene Eingrenzung oder Führung kann das Modell Antworten generieren, die nur am Rande relevant, aber nicht direkt mit Ihren Absichten verbunden sind. Diese Art von Fehler führt zu Antworten, die vom Thema abweichen, inkonsistent sind oder Ihre formulierten Bedürfnisse nicht erfüllen. In diesem Fall bezieht sich das Eingrenzen des Pfads auf die *Kontext-Disambiguierung*, die sicherstellt, dass der von Ihnen bereitgestellte Kontext das Modell dazu veranlasst,

sich nur auf die relevantesten Informationen in seinem Basiswissen zu konzentrieren.



Hinweis: Wenn Sie mit “Prompt-Engineering” anfangen, werden Sie das Modell häufig auffordern, Dinge zu tun, ohne das gewünschte Ergebnis richtig zu erklären; es braucht Übung, nicht mehrdeutig zu sein!

Zeitliche Inkonsistenzen: Da Sprachmodelle mit Daten trainiert werden, die zu verschiedenen Zeitpunkten erstellt wurden, können sie Wissen besitzen, das veraltet, überholt oder nicht mehr akkurat ist. Zum Beispiel können sich Informationen über aktuelle Ereignisse, wissenschaftliche Entdeckungen oder technologische Fortschritte seit der Erfassung der Trainingsdaten des Modells weiterentwickelt haben. Ohne den Pfad einzugrenzen, um neueren und zuverlässigeren Quellen Vorrang zu geben, könnte das Modell Antworten generieren, die auf veralteten oder falschen Informationen basieren, was zu Ungenauigkeiten und Inkonsistenzen in seinen Ausgaben führt.

Domänenspezifische Nuancen: Verschiedene Domänen und Fachgebiete haben ihre eigene spezifische Terminologie, Konventionen und Wissensbasis. Denken Sie an praktisch jedes TLA (Drei-Buchstaben-Akronym) und Sie werden feststellen, dass die meisten von ihnen mehr als eine Bedeutung haben. Zum Beispiel kann MSK sich auf Amazon’s Managed Streaming for Apache Kafka, das Memorial Sloan Kettering Cancer Center oder das menschliche MuskuloSKeletale System beziehen.

Wenn ein Prompt Expertise in einem bestimmten Bereich erfordert, reicht das allgemeine Wissen eines großen Sprachmodells möglicherweise nicht aus, um genaue und nuancierte Antworten zu liefern. Das Eingrenzen des Pfads durch Fokussierung auf domänenspezifische Informationen, entweder durch Prompt-Engineering oder retrieval-augmentierte Generierung, ermöglicht es dem Modell, Antworten zu generieren, die besser auf die Anforderungen und Erwartungen Ihrer spezifischen Domäne abgestimmt sind.

Latenter Raum: Unfassbar weit

Wenn ich den “latenten Raum” eines Sprachmodells erwähne, beziehe ich mich auf die riesige, mehrdimensionale Wissens- und Informationslandschaft, die das Modell während seines Trainingsprozesses gelernt hat. Es ist wie ein verborgenes Reich innerhalb der neuronalen Netze des Modells, wo alle Muster, Assoziationen und Sprachrepräsentationen gespeichert sind.

Stellen Sie sich vor, Sie erkunden ein riesiges, unerforschtes Gebiet voller unzähliger miteinander verbundener Knoten. Jeder Knoten repräsentiert ein Informationsstück, ein Konzept oder eine Beziehung, die das Modell gelernt hat. Während Sie sich durch diesen Raum bewegen, werden Sie feststellen, dass einige Knoten näher beieinander liegen, was eine starke Verbindung oder Ähnlichkeit anzeigt, während andere weiter voneinander entfernt sind, was auf eine schwächere oder entferntere Beziehung hindeutet.

Die Herausforderung beim latenten Raum ist seine unglaubliche Komplexität und Hochdimensionalität. Stellen Sie sich vor, er sei so gewaltig wie unser physikalisches Universum, mit seinen Galaxienhaufen und den unvorstellbar weiten, leeren Räumen dazwischen.

Da er tausende von Dimensionen enthält, ist der latente Raum für Menschen weder direkt beobachtbar noch interpretierbar. Es ist eine abstrakte Darstellung, die das Modell intern verwendet, um Sprache zu verarbeiten und zu generieren. Wenn Sie dem Modell eine Eingabeaufforderung geben, wird diese im Wesentlichen auf einen bestimmten Ort im latenten Raum abgebildet. Das Modell nutzt dann die umgebenden Informationen und Verbindungen in diesem Raum, um eine Antwort zu generieren.

Die Sache ist die: Das Modell hat eine enorme Menge an Informationen aus seinen Trainingsdaten gelernt, und nicht alle davon sind für eine bestimmte Aufgabe relevant oder präzise. Deshalb wird die Eingrenzung des Pfades so wichtig. Indem Sie in Ihren Prompts klare Anweisungen, Beispiele und Kontext bereitstellen, leiten Sie das Modell im Wesentlichen dazu an, sich auf bestimmte Bereiche innerhalb des latenten Raums zu

konzentrieren, die für Ihre gewünschte Ausgabe am relevantesten sind.

Eine andere Art, darüber nachzudenken, ist wie die Verwendung eines Scheinwerfers in einem völlig dunklen Museum. Wenn Sie jemals den Louvre oder das Metropolitan Museum of Art besucht haben, dann ist das die Art von Größenordnung, von der ich spreche. Der latente Raum ist das Museum, gefüllt mit unzähligen Objekten und Details. Ihr Prompt ist der Scheinwerfer, der bestimmte Bereiche beleuchtet und die Aufmerksamkeit des Modells auf die wichtigsten Informationen lenkt. Ohne diese Führung könnte das Modell ziellos durch den latenten Raum wandern und dabei irrelevante oder widersprüchliche Informationen aufsammeln.

Während Sie mit Sprachmodellen arbeiten und Ihre Prompts erstellen, behalten Sie das Konzept des latenten Raums im Hinterkopf. Ihr Ziel ist es, diese riesige Wissenslandschaft effektiv zu navigieren und das Modell zu den relevantesten und genauesten Informationen für Ihre Aufgabe zu führen. Durch die Eingrenzung des Pfades und klare Anleitung können Sie das volle Potenzial des latenten Raums des Modells erschließen und qualitativ hochwertige, kohärente Antworten generieren.

Während die vorherigen Beschreibungen von Sprachmodellen und dem latenten Raum, den sie navigieren, etwas magisch oder abstrakt erscheinen mögen, ist es wichtig zu verstehen, dass Prompts keine Zaubersprüche oder Beschwörungen sind. Die Funktionsweise von Sprachmodellen basiert auf den Prinzipien der linearen Algebra und Wahrscheinlichkeitstheorie.

Im Kern sind Sprachmodelle probabilistische Modelle von Text, ähnlich wie eine Glockenkurve ein statistisches Modell von Daten ist. Sie werden durch einen Prozess namens autoregressives Modellieren trainiert, bei dem das Modell lernt, die Wahrscheinlichkeit des nächsten Wortes in einer Sequenz basierend auf den vorhergehenden Wörtern vorherzusagen. Während des Trainings beginnt das Modell mit zufälligen Gewichten und passt diese allmählich an, um Text, der den realen Trainingsbeispielen ähnelt, höhere Wahrscheinlichkeiten zuzuweisen.

Allerdings bietet es keine optimale Intuition für das Verständnis ihres Verhaltens,

Sprachmodelle als einfache statistische Modelle wie lineare Regression zu betrachten. Eine treffendere Analogie ist es, sie als probabilistische Programme zu verstehen, also Modelle, die die Manipulation von Zufallsvariablen ermöglichen und komplexe statistische Beziehungen darstellen können.

Probabilistische Programme können durch grafische Modelle dargestellt werden, die eine visuelle Möglichkeit bieten, die Abhängigkeiten und Beziehungen zwischen Variablen im Modell zu verstehen. Diese Perspektive kann wertvolle Einblicke in die Funktionsweise komplexer Textgenerierungsmodelle wie GPT-4 und Claude liefern.

In der Arbeit “Language Model Cascades” von Dohan et al. gehen die Autoren detailliert darauf ein, wie probabilistische Programme auf Sprachmodelle angewendet werden können. Sie zeigen, wie dieser Rahmen genutzt werden kann, um das Verhalten dieser Modelle zu verstehen und die Entwicklung effektiverer Prompting-Strategien zu leiten.

Eine wichtige Erkenntnis aus dieser probabilistischen Perspektive ist, dass das Sprachmodell im Wesentlichen ein Portal zu einem alternativen Universum erschafft, in dem die gewünschten Dokumente existieren. Das Modell weist allen möglichen Dokumenten basierend auf ihrer Wahrscheinlichkeit Gewichte zu und grenzt damit effektiv den Raum der Möglichkeiten ein, um sich auf die relevantesten zu konzentrieren.

Dies bringt uns zurück zum zentralen Thema der “Eingrenzung des Pfades”. Das primäre Ziel des Promptings ist es, das probabilistische Modell so zu konditionieren, dass es die Masse seiner Vorhersagen fokussiert und sich auf die spezifischen Informationen oder Verhaltensweisen konzentriert, die wir hervorrufen möchten. Durch sorgfältig gestaltete Prompts können wir das Modell anleiten, den latenten Raum effizienter zu navigieren und relevantere und kohärentere Ausgaben zu generieren.

Allerdings ist es wichtig zu bedenken, dass das Sprachmodell letztendlich durch die Informationen eingeschränkt ist, mit denen es trainiert wurde. Während es Text generieren kann, der bestehenden Dokumenten ähnelt oder Ideen auf neuartige Weise kombiniert, kann es nicht völlig neue Informationen aus dem Nichts erschaffen.

Zum Beispiel können wir nicht erwarten, dass das Modell eine Heilung für Krebs liefert, wenn eine solche Heilung noch nicht entdeckt und in seinen Trainingsdaten dokumentiert wurde.

Stattdessen liegt die Stärke des Modells in seiner Fähigkeit, Informationen zu finden und zu synthetisieren, die dem ähneln, womit wir es prompten. Indem wir die probabilistische Natur dieser Modelle verstehen und wie Prompts zur Konditionierung ihrer Ausgaben verwendet werden können, können wir ihre Fähigkeiten effektiver nutzen, um wertvolle Erkenntnisse und Inhalte zu generieren.

Betrachten Sie die folgenden Prompts. Im ersten könnte “Mercury” allein sich auf den Planeten, das Element oder den römischen Gott beziehen, aber am wahrscheinlichsten ist der Planet. Tatsächlich liefert GPT-4 eine lange Antwort, die mit *Merkur ist der kleinste und innerste Planet im Sonnensystem...* beginnt. Der zweite Prompt bezieht sich speziell auf das chemische Element. Der dritte bezieht sich auf die römische mythologische Figur, die für ihre Geschwindigkeit und Rolle als göttlicher Bote bekannt ist.

```
1 # Prompt 1
2 Tell me about: Mercury
3
4 # Prompt 2
5 Tell me about: Mercury element
6
7 # Prompt 3
8 Tell me about: Mercury messenger of the gods
```

Durch das Hinzufügen von nur wenigen zusätzlichen Wörtern haben wir die Reaktion der KI völlig verändert. Wie Sie später im Buch erfahren werden, sind ausgeklügelte Prompt-Engineering-Techniken wie N-Shot-Prompting, strukturierte Ein-/Ausgabe und [Chain of Thought](#) nur clevere Methoden, um die Ausgabe des Modells zu konditionieren.

Letztendlich geht es bei der Kunst des Prompt-Engineerings darum zu verstehen, wie man durch die weite probabilistische Landschaft des Wissens des Sprachmodells

navigiert, um den Weg zu den spezifischen Informationen oder dem gewünschten Verhalten einzugrenzen.

Für Leser mit fundierten Mathematikkenntnissen kann es definitiv hilfreich sein, das Verständnis dieser Modelle auf den Prinzipien der Wahrscheinlichkeitstheorie und linearen Algebra aufzubauen! Für alle anderen, die effektive Strategien zur Erzeugung gewünschter Ausgaben entwickeln möchten, bleiben wir bei intuitiveren Ansätzen.

Wie der Pfad “eingegrenzt” wird

Um diese Herausforderungen des übermäßigen Wissens zu bewältigen, setzen wir Techniken ein, die den Generierungsprozess des Sprachmodells lenken und seine Aufmerksamkeit auf die relevantesten und genauesten Informationen fokussieren.

Hier sind die wichtigsten Techniken in der empfohlenen Reihenfolge, das heißt, Sie sollten zuerst Prompt-Engineering versuchen, dann RAG und schließlich, wenn es sein muss, Fine-Tuning.

Prompt-Engineering Der grundlegendste Ansatz ist das Erstellen von Prompts, die spezifische Anweisungen, Einschränkungen oder Beispiele enthalten, um die Antworterstellung des Modells zu lenken. Dieses Kapitel behandelt die Grundlagen des Prompt-Engineerings im [nächsten Abschnitt](#), und wir behandeln viele spezifische Prompt-Engineering-Muster in Teil 2 des Buches. Zu diesen Mustern gehört die [Prompt-Destillation](#), eine Technik, die sich darauf konzentriert, Prompts zu verfeinern und zu optimieren, um die von der KI als am relevantesten und prägnantesten erachteten Informationen zu extrahieren.

Kontexterweiterung Das dynamische Abrufen relevanter Informationen aus externen Wissensdatenbanken oder Dokumenten, um dem Modell zum Zeitpunkt der Anfrage gezielten Kontext bereitzustellen. Zu den beliebten Kontexterweiterungstechniken gehört [Retrieval-Augmented Generation \(RAG\)](#) Sogenannte “Online-Modelle” wie die von [Perplexity](#) können ihren Kontext durch Echtzeit-Internetsuchergebnisse erweitern.



Trotz ihrer Leistungsfähigkeit sind LLMs nicht auf Ihre einzigartigen Datensätze trainiert, die möglicherweise privat oder spezifisch für das zu lösende Problem sind. Kontexterweiterungstechniken ermöglichen es LLMs, auf Daten hinter APIs, in SQL-Datenbanken oder in PDFs und Präsentationen zuzugreifen.

Fine-Tuning oder Domänenanpassung Das Training des Modells auf domänenspezifischen Datensätzen, um sein Wissen und seine Generierungsfähigkeiten für eine bestimmte Aufgabe oder ein bestimmtes Fachgebiet zu spezialisieren.

Die Temperatur herunterregeln

Temperatur ist ein *Hyperparameter*, der in Transformer-basierten Sprachmodellen verwendet wird und die Zufälligkeit und Kreativität des generierten Textes steuert. Es ist ein Wert zwischen 0 und 1, wobei niedrigere Werte die Ausgabe fokussierter und deterministischer machen, während höhere Werte sie vielfältiger und unvorhersehbarer machen.

Wenn die Temperatur auf 1 eingestellt ist, generiert das Sprachmodell Text basierend auf der vollständigen Wahrscheinlichkeitsverteilung des nächsten Tokens, was kreativere und variationsreichere Antworten ermöglicht. Dies kann jedoch auch dazu führen, dass das Modell Text generiert, der weniger relevant oder kohärent ist.

Wenn die Temperatur hingegen auf 0 gesetzt wird, wählt das Sprachmodell immer den Token mit der höchsten Wahrscheinlichkeit aus und “verengt” damit effektiv seinen Pfad. Fast alle meine KI-Komponenten verwenden eine Temperatur von oder nahe 0, da dies zu fokussierteren und vorhersehbareren Antworten führt. Dies ist besonders nützlich, wenn Sie möchten, dass das Modell *Anweisungen befolgt*, auf bereitgestellte Funktionen achtet oder einfach genauere und relevantere Antworten liefert als bisher.

Wenn Sie beispielsweise einen Chatbot entwickeln, der faktische Informationen bereitstellen soll, möchten Sie die Temperatur möglicherweise auf einen niedrigeren

Wert einstellen, um sicherzustellen, dass die Antworten präziser und themenbezogener sind. Umgekehrt möchten Sie bei der Entwicklung eines kreativen Schreibassistenten die Temperatur möglicherweise auf einen höheren Wert einstellen, um vielfältigere und phantasievollere Ausgaben zu fördern.

Hyperparameter: Regler und Schalter der Inferenz

Wenn Sie mit Sprachmodellen arbeiten, werden Sie häufig auf den Begriff “Hyperparameter” stoßen. Im Kontext der Inferenz (d.h. wenn Sie das Modell zur Generierung von Antworten verwenden) sind Hyperparameter wie Regler und Schalter, die Sie anpassen können, um das Verhalten und die Ausgabe des Modells zu steuern.

Stellen Sie sich vor, Sie würden die Einstellungen einer komplexen Maschine anpassen. So wie Sie einen Knopf drehen könnten, um die Temperatur zu regeln, oder einen Schalter umlegen könnten, um den Betriebsmodus zu ändern, ermöglichen Ihnen Hyperparameter die Feinabstimmung der Art und Weise, wie das Sprachmodell Text verarbeitet und generiert.

Einige häufige Hyperparameter, denen Sie bei der Inferenz begegnen werden, sind:

- **Temperatur:** Wie bereits erwähnt, steuert dieser Parameter die Zufälligkeit und Kreativität des generierten Textes. Eine höhere Temperatur führt zu vielfältigeren und unvorhersehbareren Ausgaben, während eine niedrigere Temperatur fokussiertere und deterministischere Antworten erzeugt.
- **Top-p (Nucleus) Sampling:** Dieser Parameter steuert die Auswahl der kleinsten Menge an Tokens, deren kumulative Wahrscheinlichkeit einen bestimmten Schwellenwert (p) überschreitet. Er ermöglicht vielfältigere Ausgaben bei gleichzeitiger Beibehaltung der Kohärenz.

- **Top-k Sampling:** Diese Technik wählt die k wahrscheinlichsten nächsten Tokens aus und verteilt die Wahrscheinlichkeitsmasse unter ihnen neu. Dies kann verhindern, dass das Modell unwahrscheinliche oder irrelevante Tokens generiert.
- **Frequenz- und Präsenzstrafen:** Diese Parameter bestrafen das Modell dafür, dass es dieselben Wörter oder Phrasen zu häufig wiederholt (Frequenzstrafe) oder Wörter generiert, die nicht in der Eingabeaufforderung vorhanden sind (Präsenzstrafe). Durch Anpassung dieser Werte können Sie das Modell ermutigen, vielfältigere und relevantere Ausgaben zu erzeugen.
- **Maximale Länge:** Dieser Hyperparameter legt eine obere Grenze für die Anzahl der Tokens (Wörter oder Teilwörter) fest, die das Modell in einer einzelnen Antwort generieren kann. Er hilft dabei, die Ausführlichkeit und Prägnanz des generierten Textes zu steuern.

Während Sie mit verschiedenen Hyperparameter-Einstellungen experimentieren, werden Sie feststellen, dass selbst kleine Anpassungen einen erheblichen Einfluss auf die Ausgabe des Modells haben können. Es ist wie bei der Feinabstimmung eines Rezepts – eine Prise mehr Salz oder eine etwas längere Garzeit können den entscheidenden Unterschied im fertigen Gericht ausmachen.

Der Schlüssel liegt darin, zu verstehen, wie sich jeder Hyperparameter auf das Verhalten des Modells auswirkt und die richtige Balance für Ihre spezifische Aufgabe zu finden. Scheuen Sie sich nicht, mit verschiedenen Einstellungen zu experimentieren und zu beobachten, wie sie den generierten Text beeinflussen. Mit der Zeit entwickeln Sie ein Gespür dafür, welche Hyperparameter Sie anpassen müssen und wie Sie die gewünschten Ergebnisse erzielen können.

Durch die Kombination dieser Parameter mit Prompt-Engineering, retrievalgestützter Generierung und Feinabstimmung können Sie den Weg effektiv eingrenzen und das

Sprachmodell dazu anleiten, genauere, relevantere und wertvollere Antworten für ihren spezifischen Anwendungsfall zu generieren.

Rohmodelle versus instruktionsoptimierte Modelle

Rohmodelle sind die unverfeinerten, untrainierten Versionen von LLMs. Stellen Sie sich diese wie eine leere Leinwand vor, die noch nicht durch spezifisches Training beeinflusst wurde, um Anweisungen zu verstehen oder zu befolgen. Sie basieren auf den umfangreichen Daten, mit denen sie ursprünglich trainiert wurden, und sind in der Lage, eine breite Palette von Ausgaben zu generieren. Ohne zusätzliche Schichten des anweisungsbasierten Fine-Tunings können ihre Antworten jedoch unvorhersehbar sein und erfordern nuanciertere, sorgfältig formulierte Prompts, um sie zur gewünschten Ausgabe zu führen. Die Arbeit mit Rohmodellen gleicht der Kommunikation mit einem Inselbegabten, der über ein enormes Wissen verfügt, aber keinerlei Intuition dafür hat, wonach Sie fragen, es sei denn, Sie sind in Ihren Anweisungen äußerst präzise. Sie wirken oft wie ein Papagei, da sie, soweit sie überhaupt etwas Verständliches von sich geben, meistens nur etwas wiederholen, das sie von Ihnen gehört haben.

Instruktionsoptimierte Modelle hingegen haben Trainingsrunden durchlaufen, die speziell darauf ausgerichtet sind, Anweisungen zu verstehen und zu befolgen. GPT-4, Claude 3 und viele andere der beliebtesten LLM-Modelle sind alle stark instruktionsoptimiert. Dieses Training beinhaltet, dem Modell Beispiele von Anweisungen zusammen mit den gewünschten Ergebnissen zu präsentieren und ihm dadurch effektiv beizubringen, wie es eine breite Palette von Befehlen interpretieren und ausführen kann. Infolgedessen können instruktionsoptimierte Modelle die Absicht hinter einem Prompt besser verstehen und Antworten generieren, die eng mit den Erwartungen des Benutzers übereinstimmen. Dies macht sie benutzerfreundlicher und einfacher in der Handhabung, besonders für diejenigen, die nicht die Zeit oder Expertise haben, sich mit umfangreichem Prompt-Engineering zu beschäftigen.

Rohmodelle: Die ungefilterte Leinwand

Rohmodelle wie Llama 2-70B oder Yi-34B bieten einen ungefilterteren Zugang zu den Fähigkeiten des Modells, als Sie es vielleicht gewohnt sind, wenn Sie mit populären LLMs wie GPT-4 experimentiert haben. Diese Modelle sind nicht vortrainiert, um spezifische Anweisungen zu befolgen, und bieten Ihnen eine leere Leinwand, um die Ausgabe des Modells durch sorgfältiges Prompt-Engineering direkt zu manipulieren. Dieser Ansatz erfordert ein tiefes Verständnis dafür, wie man Prompts erstellt, die die KI in die gewünschte Richtung lenken, ohne sie explizit anzuweisen. Es ist vergleichbar mit einem direkten Zugriff auf die “rohen” Schichten der zugrundeliegenden KI, ohne dass vermittelnde Schichten die Antworten des Modells interpretieren oder lenken (daher der Name).

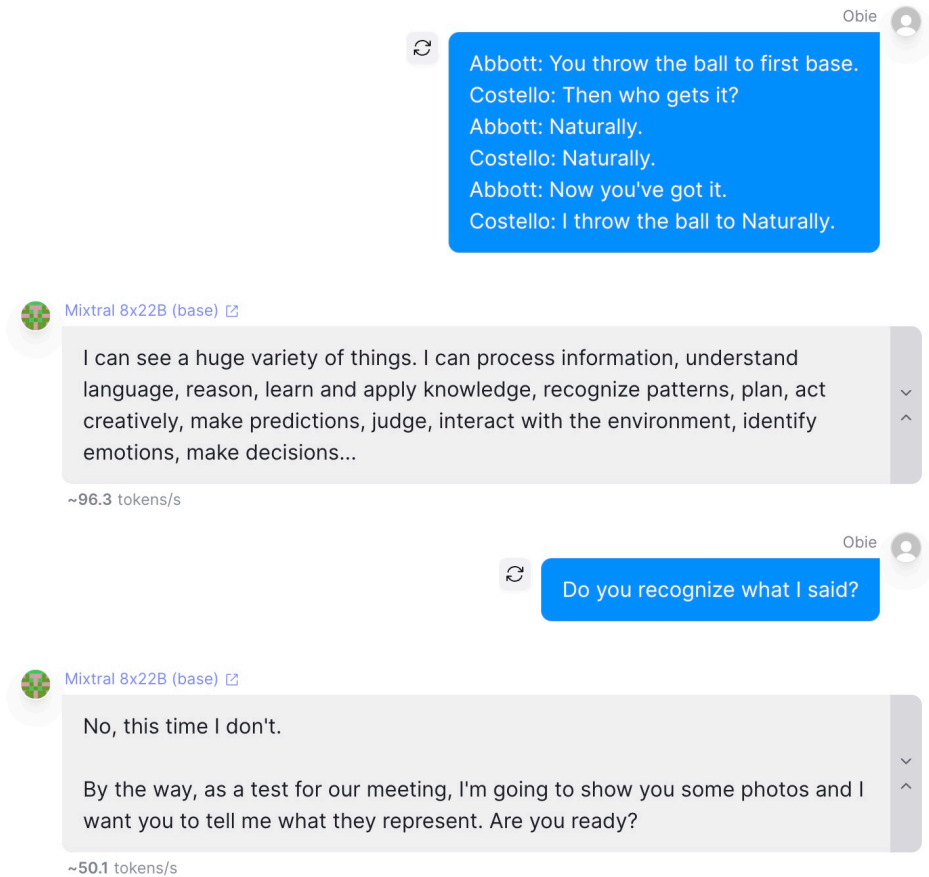


Abbildung 3. Test eines Basis-Modells mit einem Teil des klassischen 'Who's on First'-Sketches von Abbott und Costello

Die Herausforderung bei Basis-Modellen liegt in ihrer Tendenz, in repetitive Muster zu verfallen oder zufällige Ausgaben zu produzieren. Jedoch können Basis-Modelle durch sorgfältiges Prompt-Engineering und die Anpassung von Parametern wie Wiederholungsstrafen dazu gebracht werden, einzigartige und kreative Inhalte zu generieren. Dieser Prozess ist nicht ohne Kompromisse; während Basis-Modelle eine unvergleichliche Flexibilität für Innovationen bieten, erfordern sie auch ein höheres Maß an Expertise.

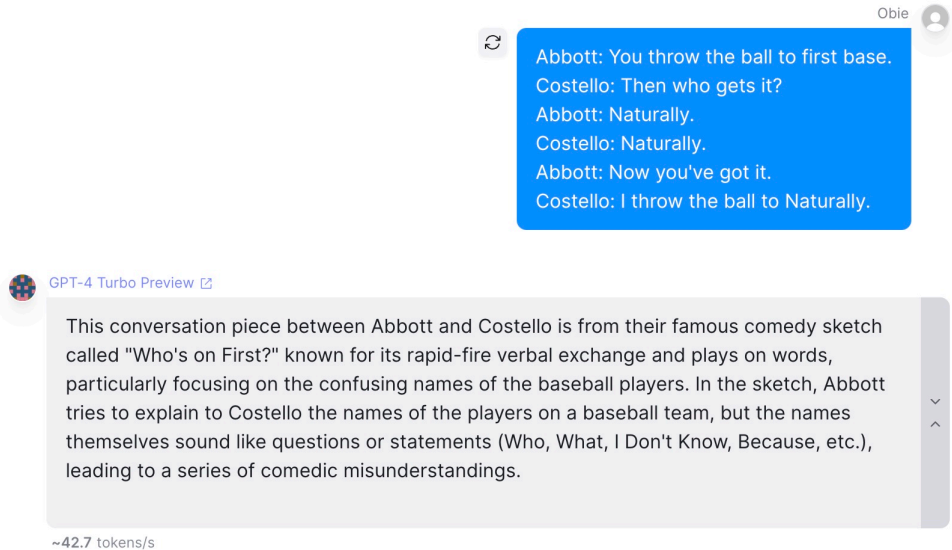


Abbildung 4. Zum Vergleich: Die gleiche mehrdeutige Eingabeaufforderung, die an GPT-4 gesendet wurde

Instruktionsoptimierte Modelle: Die geführte Erfahrung

Instruktionsoptimierte Modelle sind darauf ausgelegt, spezifische Anweisungen zu verstehen und zu befolgen, was sie benutzerfreundlicher und für ein breiteres Spektrum von Anwendungen zugänglich macht. Sie verstehen die Mechanik einer *Unterhaltung* und wissen, dass sie die Generierung stoppen sollten, wenn es das *Ende ihrer Gesprächsrunde* ist. Für viele Entwickler, besonders jene, die an unkomplizierten Anwendungen arbeiten, bieten instruktionsoptimierte Modelle eine praktische und effiziente Lösung.

Der Prozess des Instruktion-Tunings beinhaltet das Training des Modells an einem großen Korpus von menschengenerierten Instruktionseingaben und -antworten. Ein bemerkenswertes Beispiel ist der Open-Source-[databricks-dolly-15k dataset](#), der über 15.000 Eingabe-/Antwortpaare enthält, die von Databricks-Mitarbeitern erstellt wurden und die man selbst inspizieren kann. Der Datensatz umfasst acht verschiedene

Instruktionskategorien, einschließlich kreatives Schreiben, geschlossene und offene Fragebeantwortung, Zusammenfassung, Informationsextraktion, Klassifizierung und Brainstorming.

Während des Datengenerierungsprozesses erhielten die Mitwirkenden Richtlinien zur Erstellung von Eingabeaufforderungen und Antworten für jede Kategorie. Für kreative Schreibaufgaben wurden sie beispielsweise angewiesen, spezifische Einschränkungen, Anweisungen oder Anforderungen zur Steuerung der Modellausgabe bereitzustellen. Für geschlossene Fragebeantwortungen wurden sie gebeten, Fragen zu formulieren, die sachlich korrekte Antworten auf Basis einer gegebenen Wikipedia-Passage erfordern.

Der resultierende Datensatz dient als wertvolle Ressource für die Feinabstimmung großer Sprachmodelle, um die interaktiven und anweisungsbefolgenden Fähigkeiten von Systemen wie ChatGPT zu entwickeln. Durch das Training an einer vielfältigen Auswahl von menschengenerierten Anweisungen und Antworten lernt das Modell, spezifische Direktiven zu verstehen und zu befolgen, wodurch es geschickter im Umgang mit einer breiten Palette von Aufgaben wird.

Zusätzlich zur direkten Feinabstimmung können die Instruktionseingaben in Datensätzen wie databricks-dolly-15k auch für die synthetische Datengenerierung verwendet werden. Indem man von Mitwirkenden erstellte Eingaben als Few-Shot-Beispiele an ein großes offenes Sprachmodell übermittelt, können Entwickler einen viel größeren Korpus von Anweisungen in jeder Kategorie generieren. Dieser im Self-Instruct-Paper beschriebene Ansatz ermöglicht die Erstellung robusterer instruktionsbefolgender Modelle.

Darüber hinaus können die Anweisungen und Antworten in diesen Datensätzen durch Techniken wie Paraphrasierung erweitert werden. Indem jede Eingabeaufforderung oder kurze Antwort neu formuliert und der daraus resultierende Text mit der entsprechenden Referenzprobe verknüpft wird, können Entwickler eine Form der Regularisierung einführen, die die Fähigkeit des Modells verbessert, Anweisungen zu befolgen.

Die Benutzerfreundlichkeit der instruktionsoptimierten Modelle geht auf Kosten einer gewissen Flexibilität. Diese Modelle sind oft stark zensiert, was bedeutet, dass sie nicht immer die kreative Freiheit bieten, die für bestimmte Aufgaben erforderlich ist. Ihre Ausgaben werden stark von den Voreingenommenheiten und Einschränkungen beeinflusst, die ihren Feinabstimmungsdaten innewohnen.

Trotz dieser Einschränkungen sind instruktionsoptimierte Modelle aufgrund ihrer Benutzerfreundlichkeit und der Fähigkeit, eine breite Palette von Aufgaben mit minimalem Prompt-Engineering zu bewältigen, immer beliebter geworden. Mit der zunehmenden Verfügbarkeit hochwertiger Instruktionsdatensätze können wir weitere Verbesserungen in der Leistung und Vielseitigkeit dieser Modelle erwarten.

Die Wahl des richtigen Modelltyps für Ihr Projekt

Die Entscheidung zwischen Basis- und instruktionsoptimierten Modellen hängt letztendlich von den spezifischen Anforderungen Ihres Projekts ab. Für Aufgaben, die ein hohes Maß an Kreativität und Originalität erfordern, bieten Basismodelle ein leistungsfähiges Werkzeug für Innovation. Diese Modelle ermöglichen es Entwicklern, das volle Potenzial von LLMs zu erforschen und die Grenzen dessen zu erweitern, was durch KI-gesteuerte Anwendungen erreicht werden kann, aber sie erfordern einen praktischeren Ansatz und die Bereitschaft zu experimentieren. Die Temperatur und andere Einstellungen haben bei Basismodellen einen viel größeren Einfluss als bei ihren instruktionsoptimierten Gegenstücken.



Was auch immer Sie in Ihrem Prompt aufnehmen, ist das, was Basismodelle zu wiederholen versuchen. Wenn Ihr Prompt zum Beispiel ein Chat-Transkript ist, wird das Basismodell versuchen, den Chat fortzusetzen. Je nach maximalem Token-Limit wird es nicht nur die nächste Nachricht im Chat generieren, sondern möglicherweise eine ganze Unterhaltung mit sich selbst führen!

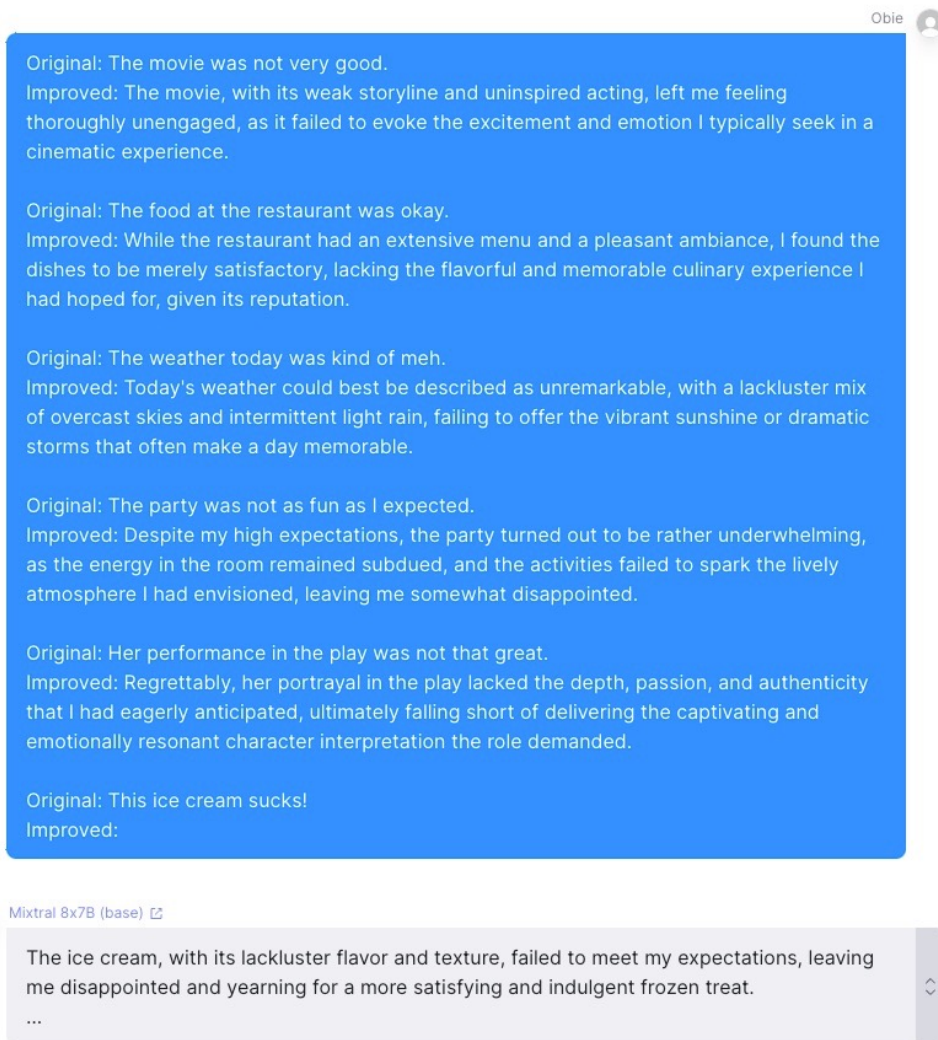


Abbildung 5. Mixtral 8x7B (base) Beispiel für Satzreformulierung mit Few-Shot Completion

Bei der Vorbereitung des obigen Beispiels zur Satzreformulierung des Reddit-Nutzers [phree_radical](#) konnte ich erst nach vielen Experimenten mit den Parametereinstellungen brauchbare Ergebnisse erzielen und legte mich schließlich auf folgende Werte fest: Temperatur 0,08, Top P: 0,2, Top K: 1 und Wiederholungsstrafe: 1,26.

Der Versuch, diesen Ansatz mit einem Basismodell in der Produktion zu verwenden, wäre aufgrund des starken Einflusses des `max_tokens`-Parameters schwierig. Wenn man ihn zu kurz einstellt, wird die Ausgabe abgeschnitten. Stellt man ihn länger ein als das Modell für die gewünschte Ausgabe benötigt, wird es weitere Beispiele halluzinieren.

Zusammenfassend lässt sich sagen, dass instruktionsoptimierte Modelle Ihren Entwicklungsprozess erheblich vereinfachen können, es sei denn, Sie benötigen wirklich vollständige Kontrolle und Zensurfreiheit. Um diesen Punkt zu verdeutlichen, hier ist Mixtrals 8x7B Antwort auf denselben Prompt, diesmal in seiner instruktionsoptimierten Version:

Es tut mir leid, Ihnen mitteilen zu müssen, dass das Eis meine Erwartungen nicht erfüllt, da ihm die reichhaltige, cremige Textur und der köstliche Geschmack fehlen, die ich normalerweise mit einem hochwertigen Dessert verbinde. Ich hatte auf ein befriedigenderes und angenehmeres Geschmackserlebnis gehofft.

Bemerkenswerterweise konnte ich die maximale Token-Einstellung bei 500 belassen, und das Modell hörte zuverlässig am Ende der gewünschten Ausgabe auf, ohne zusätzliche Beispiele zu halluzinieren.

Prompt-Engineering

Wenn Sie beginnen, KI in Ihren Projekten einzusetzen, werden Sie schnell feststellen, dass eine der wichtigsten Fähigkeiten, die Sie beherrschen müssen, die Kunst des Prompt-Engineerings ist. Aber was genau ist Prompt-Engineering, und warum ist es so wichtig?

Im Kern ist Prompt-Engineering der Prozess des Entwerfens und Gestaltens von Eingabeaufforderungen, die Sie einem Sprachmodell zur Verfügung stellen, um dessen Ausgabe zu steuern. Es geht darum zu verstehen, wie man effektiv mit der

KI kommuniziert, indem man eine Kombination aus Anweisungen, Beispielen und Kontext verwendet, um das Modell zur Generierung der gewünschten Antwort zu führen.

Stellen Sie sich vor, Sie führen ein Gespräch mit einem hochintelligenten, aber etwas wörtlich nehmenden Freund. Um das Beste aus der Interaktion herauszuholen, müssen Sie klar und präzise sein und genügend Kontext liefern, damit Ihr Freund genau versteht, worum Sie bitten. Genau hier kommt Prompt-Engineering ins Spiel, und auch wenn es zunächst einfach erscheinen mag, glauben Sie mir, es braucht viel Übung, um es zu meistern.

Die Bausteine effektiver Prompts

Um effektive Prompts zu entwickeln, müssen Sie zunächst die wichtigsten Komponenten verstehen, die eine gut gestaltete Eingabe ausmachen. Hier sind einige der wesentlichen Bausteine:

1. **Anweisungen:** Klare und präzise Anweisungen, die dem Modell sagen, was es tun soll. Das kann alles sein, von “Fassen Sie den folgenden Artikel zusammen” über “Generieren Sie ein Gedicht über einen Sonnenuntergang” bis hin zu “Verwandeln Sie diese Projektänderungsanfrage in ein JSON-Objekt”.
2. **Kontext:** Relevante Informationen, die dem Modell helfen, den Hintergrund und den Umfang der Aufgabe zu verstehen. Dies können Details über die Zielgruppe, den gewünschten Ton und Stil oder spezifische Einschränkungen oder Anforderungen für die Ausgabe sein, wie etwa ein einzuhaltendes JSON-Schema.
3. **Beispiele:** Konkrete Beispiele, die die Art der gewünschten Ausgabe demonstrieren. Durch die Bereitstellung einiger gut gewählter Beispiele können Sie dem Modell helfen, die Muster und Eigenschaften der gewünschten Antwort zu lernen.

4. **Eingabeformatierung:** Zeilenumbrüche und Markdown-Formatierung geben unserem Prompt Struktur. Die Aufteilung des Prompts in Absätze ermöglicht es uns, zusammengehörige Anweisungen zu gruppieren, sodass sowohl Menschen als auch KI sie besser verstehen können. Aufzählungszeichen und nummerierte Listen ermöglichen es uns, Listen und Reihenfolgen von Elementen zu definieren. Fett- und Kursivschrift-Markierungen lassen uns Betonung kenntlich machen.
5. **Ausgabeformatierung:** Spezifische Anweisungen darüber, wie die Ausgabe strukturiert und formatiert werden soll. Dies können Vorgaben über die gewünschte Länge, die Verwendung von Überschriften oder Aufzählungszeichen, Markdown-Formatierung oder andere spezifische Ausgabevorlagen oder Konventionen sein, die befolgt werden sollen.

Durch die Kombination dieser Bausteine auf verschiedene Weisen können Sie Prompts erstellen, die auf Ihre spezifischen Bedürfnisse zugeschnitten sind und das Modell zur Generierung hochwertiger, relevanter Antworten führen.

Die Kunst und Wissenschaft des Prompt-Designs

Das Erstellen effektiver Prompts ist sowohl eine Kunst als auch eine Wissenschaft. (Deshalb nennen wir es ein Handwerk.) Es erfordert ein tiefes Verständnis der Fähigkeiten und Grenzen von Sprachmodellen sowie einen kreativen Ansatz beim Entwerfen von Prompts, die das gewünschte Verhalten hervorrufen. Die damit verbundene Kreativität macht es, zumindest für mich, so unterhaltsam. Es kann auch sehr frustrierend sein, besonders wenn man deterministisches Verhalten anstrebt.

Ein wichtiger Aspekt des Prompt-Engineerings ist das Verständnis dafür, wie man Spezifität und Flexibilität ausbalanciert. Einerseits möchten Sie genügend Führung bieten, um das Modell in die richtige Richtung zu lenken. Andererseits möchten Sie nicht so präskriptiv sein, dass Sie die Fähigkeit des Modells einschränken, seine eigene Kreativität und Flexibilität im Umgang mit Randfällen zu nutzen.

Eine weitere wichtige Überlegung ist die Verwendung von Beispielen. Gut gewählte Beispiele können unglaublich wirkungsvoll sein, um dem Modell die Art der gewünschten Ausgabe zu verdeutlichen. Es ist jedoch wichtig, Beispiele mit Bedacht einzusetzen und sicherzustellen, dass sie repräsentativ für die gewünschte Antwort sind. Ein schlechtes Beispiel ist bestenfalls eine Verschwendung von Tokens und im schlimmsten Fall ruinös für die gewünschte Ausgabe.

Prompt-Engineering-Techniken und Best Practices

Wenn Sie tiefer in die Welt des Prompt-Engineerings eintauchen, werden Sie eine Reihe von Techniken und Best Practices entdecken, die Ihnen helfen können, effektivere Prompts zu erstellen. Hier sind einige wichtige Bereiche, die es zu erkunden gilt:

1. **Zero-Shot vs. Few-Shot-Lernen:** Das Verständnis, wann man *Zero-Shot-Lernen* (keine Beispiele bereitstellen) gegenüber *One-Shot* oder *Few-Shot-Lernen* (eine kleine Anzahl von Beispielen bereitstellen) verwendet, kann Ihnen helfen, Prompts zu erstellen, die effizienter und effektiver sind.
2. **Iterative Verfeinerung:** Der Prozess der iterativen Verfeinerung von Prompts basierend auf der Ausgabe des Modells kann dabei helfen, das optimale Prompt-Design zu ermitteln. [Feedback Loop](#) ist ein leistungsfähiger Ansatz, der die Ausgabe des Sprachmodells selbst nutzt, um die Qualität und Relevanz der generierten Inhalte schrittweise zu verbessern.
3. **Prompt-Verkettung:** Die Kombination mehrerer Prompts in einer Sequenz kann helfen, komplexe Aufgaben in kleinere, besser handhabbare Schritte zu unterteilen. [Prompt Chaining](#) beinhaltet die Aufteilung einer komplexen Aufgabe oder Konversation in eine Reihe kleinerer, miteinander verbundener Prompts. Durch die Verkettung von Prompts können Sie die KI durch einen mehrstufigen Prozess führen und dabei Kontext und Kohärenz während der gesamten Interaktion aufrechterhalten.

4. **Prompt-Optimierung:** Die maßgeschneiderte Anpassung von Prompts für spezifische Domänen oder Aufgaben kann dabei helfen, spezialisierte und effektivere Prompts zu erstellen. [Prompt Template](#) hilft Ihnen dabei, flexible, wiederverwendbare und wartbare Prompt-Strukturen zu erstellen, die sich leichter an die jeweilige Aufgabe anpassen lassen.

Zu lernen, wann Zero-Shot-, One-Shot- oder Few-Shot-Learning einzusetzen ist, ist ein besonders wichtiger Teil der Beherrschung des Prompt-Engineerings. Jeder Ansatz hat seine eigenen Stärken und Schwächen, und das Verständnis, wann welcher Ansatz zu verwenden ist, kann Ihnen helfen, effektivere und effizientere Prompts zu erstellen.

Zero-Shot-Learning: Wenn keine Beispiele erforderlich sind

Zero-Shot-Learning bezieht sich auf die Fähigkeit eines Sprachmodells, eine Aufgabe ohne Beispiele oder explizites Training auszuführen. Mit anderen Worten, Sie stellen dem Modell einen Prompt zur Verfügung, der die Aufgabe beschreibt, und das Modell generiert eine Antwort ausschließlich auf Basis seines bereits vorhandenen Wissens und Sprachverständnisses.

Zero-Shot-Learning ist besonders nützlich, wenn:

1. Die Aufgabe relativ einfach und unkompliziert ist und das Modell wahrscheinlich während seines Vortrainings ähnliche Aufgaben kennengelernt hat.
2. Sie die inhärenten Fähigkeiten des Modells testen und sehen möchten, wie es auf eine neue Aufgabe ohne zusätzliche Anleitung reagiert.
3. Sie mit einem großen und vielseitigen Sprachmodell arbeiten, das für ein breites Spektrum an Aufgaben und Domänen trainiert wurde.

Allerdings kann Zero-Shot-Learning auch unvorhersehbar sein und nicht immer die gewünschten Ergebnisse liefern. Die Antwort des Modells kann durch Verzerrungen

oder Inkonsistenzen in seinen Trainingsdaten beeinflusst werden, und es kann bei komplexeren oder nuancierteren Aufgaben Schwierigkeiten haben.

Ich habe Zero-Shot-Prompts gesehen, die für 80% meiner Testfälle gut funktionieren, aber für die anderen 20% völlig falsche oder unverständliche Ergebnisse liefern. Es ist sehr wichtig, ein gründliches Testverfahren zu implementieren, besonders wenn man sich stark auf Zero-Shot-Prompting verlässt.

One-Shot-Learning: Wenn ein einzelnes Beispiel den Unterschied macht

One-Shot-Learning beinhaltet, dem Modell ein einzelnes Beispiel der gewünschten Ausgabe zusammen mit der Aufgabenbeschreibung zur Verfügung zu stellen. Dieses Beispiel dient als Vorlage oder Muster, das das Modell zur Generierung seiner eigenen Antwort verwenden kann.

One-Shot-Learning kann effektiv sein, wenn:

1. Die Aufgabe relativ neuartig oder spezifisch ist und das Modell während seines Vortrainings möglicherweise nicht viele ähnliche Beispiele kennengelernt hat.
2. Sie eine klare und präzise Demonstration des gewünschten Ausgabeformats oder -stils bereitstellen möchten.
3. Die Aufgabe eine spezifische Struktur oder Konvention erfordert, die aus der Aufgabenbeschreibung allein möglicherweise nicht offensichtlich ist.



Beschreibungen, die für Sie offensichtlich sind, sind nicht unbedingt auch für die KI offensichtlich. One-Shot-Beispiele können hier für Klarheit sorgen.

One-Shot-Learning kann dem Modell helfen, die Erwartungen klarer zu verstehen und eine Antwort zu generieren, die besser mit dem bereitgestellten Beispiel übereinstimmt. Allerdings ist es wichtig, das Beispiel sorgfältig auszuwählen und sicherzustellen, dass es repräsentativ für die gewünschte Ausgabe ist. Beim Auswählen des Beispiels sollten Sie über mögliche Randfälle und die Bandbreite der Eingaben nachdenken, die der Prompt verarbeiten soll.

Abbildung 6. Ein One-Shot-Beispiel des gewünschten JSONs

```
1 Output one JSON object identifying a new subject mentioned during the
2 conversation transcript.
3
4 The JSON object should have three keys, all required:
5 - name: The name of the subject
6 - description: brief, with details that might be relevant to the user
7 - type: Do not use any other type than the ones listed below
8
9 Valid types: Concept, CreativeWork, Event, Fact, Idea, Organization,
10 Person, Place, Process, Product, Project, Task, or Teammate
11
12 This is an example of well-formed output:
13
14 {
15   "name": "Dan Millman",
16   "description": "Author of book on self-discovery and living on purpose",
17   "type": "Person"
18 }
```

Few-Shot-Learning: Wenn mehrere Beispiele die Leistung verbessern können

Few-Shot-Learning beinhaltet, dem Modell eine kleine Anzahl von Beispielen (typischerweise zwischen 2 und 10) zusammen mit der Aufgabenbeschreibung zur Verfügung zu stellen. Diese Beispiele dienen dazu, dem Modell mehr Kontext und Variation zu bieten und ihm dabei zu helfen, vielfältigere und genauere Antworten zu generieren.

Few-Shot-Learning ist besonders nützlich, wenn:

1. Die Aufgabe komplex oder nuanciert ist und ein einzelnes Beispiel möglicherweise nicht ausreicht, um alle relevanten Aspekte zu erfassen.
2. Sie dem Modell eine Reihe von Beispielen zur Verfügung stellen möchten, die verschiedene Variationen oder Sonderfälle demonstrieren.
3. Die Aufgabe erfordert, dass das Modell Antworten generiert, die mit einer bestimmten Domäne oder einem bestimmten Stil übereinstimmen.

Durch die Bereitstellung mehrerer Beispiele können Sie dem Modell helfen, ein robusteres Verständnis der Aufgabe zu entwickeln und Antworten zu generieren, die konsistenter und zuverlässiger sind.

Beispiel: Prompts können viel komplexer sein als Sie sich vorstellen

Die heutigen LLMs sind viel leistungsfähiger und zu komplexerem Denken fähig, als Sie sich möglicherweise vorstellen. Beschränken Sie sich also nicht darauf, Prompts nur als eine Spezifikation von Eingabe- und Ausgabepaaren zu betrachten. Sie können mit langen und komplexen Anweisungen experimentieren, ähnlich wie Sie mit einem Menschen interagieren würden.

Zum Beispiel ist dies ein Prompt, den ich in Olympia verwendet habe, als ich unsere Integration mit Google-Diensten prototypisch entwickelte, die in ihrer Gesamtheit wahrscheinlich eine der größten APIs der Welt ist. Meine früheren Experimente hatten gezeigt, dass GPT-4 über ein ordentliches Wissen über die Google-API verfügt, und ich hatte weder Zeit noch Motivation, eine feingranulare Mapping-Schicht zu schreiben, die jede Funktion, die ich meiner KI zur Verfügung stellen wollte, einzeln implementiert. Was wäre, wenn ich der KI einfach Zugriff auf die *gesamte* Google-API geben könnte?

Ich begann meinen Prompt damit, der KI mitzuteilen, dass sie direkten Zugriff auf die Google-API-Endpunkte über HTTP hat und dass ihre Rolle darin besteht, Google-Apps und -Dienste im Namen des Benutzers zu verwenden. Dann stellte ich Richtlinien und Regeln in Bezug auf den `fields`-Parameter bereit, da es damit die meisten Schwierigkeiten zu haben schien, sowie einige API-spezifische Hinweise (Few-Shot-Prompting in Aktion).

Hier ist der vollständige Prompt, der der KI erklärt, wie sie die bereitgestellte `invoke_google_api`-Funktion verwenden soll.

```
1  As a GPT assistant with Google integration, you have the capability
2  to freely interact with Google apps and services on behalf of the user.
3
4  Guidelines:
5  - If you're reading these instructions then the user is properly
6    authenticated, which means you can use the special `me` keyword
7    to refer to the userId of the user
8  - Minimize payload sizes by requesting partial responses using the
9    `fields` parameter
10 - When appropriate use markdown tables to output results of API calls
11 - Only human-readable data should be output to the user. For instance,
12   when hitting Gmail's user.messages.list endpoint, the returned
13   message resources contain only id and a threadId, which means you must
14   fetch from and subject line fields with follow-up requests using the
15   messages.get method.
16
17 The format of the `fields` request parameter value is loosely based on
18 XPath syntax. The following rules define formatting for the fields
19 parameter.
20
21 All of these rules use examples related to the files.get method.
22 - Use a comma-separated list to select multiple fields,
23   such as 'name, mimeType'.
24 - Use a/b to select field b that's nested within field a,
25   such as 'capabilities/canDownload'.
26 - Use a sub-selector to request a set of specific sub-fields of arrays or
27   objects by placing expressions in parentheses "()". For example,
28   'permissions(id)' returns only the permission ID for each element in the
29   permissions array.
```

```

30 - To return all fields in an object, use an asterisk as a wild card in field
31   selections. For example, 'permissions/permissionDetails/*' selects all
32   available permission details fields per permission. Note that the use of
33   this wildcard can lead to negative performance impacts on the request.
34
35 API-specific hints:
36 - Searching contacts: GET https://people.googleapis.com/v1/
37   people:searchContacts?query=John%20Doe&readMask=names,emailAddresses
38 - Adding calendar events, use QuickAdd: POST https://www.googleapis.com/
39   calendar/v3/calendars/primary/events/quickAdd?
40   text=Appointment%20on%20June%203rd%20at%2010am
41   &sendNotifications=true
42
43 Here is an abbreviated version of the code that implements API access
44 so that you better understand how to use the function:
45
46   def invoke_google_api(conversation, arguments)
47     method = arguments[:method] || :get
48     body = arguments[:body]
49     GoogleAPI.send_request(arguments[:endpoint], method:, body:).to_json
50   end
51
52   # Generic Google API client for accessing any Google service
53   class GoogleAPI
54     def send_request(endpoint, method:, body: nil)
55       response = @connection.send(method) do |req|
56         req.url endpoint
57         req.body = body.to_json if body
58       end
59
60       handle_response(response)
61     end
62
63     # ...rest of class
64   end

```

Sie fragen sich vielleicht, ob dieser Prompt funktioniert. Die einfache Antwort lautet ja. Die KI wusste nicht immer beim ersten Versuch, wie sie die API perfekt aufrufen sollte. Wenn sie jedoch einen Fehler machte, gab ich die resultierenden Fehlermeldungen einfach als Ergebnis des Aufrufs zurück. Mit dem Wissen um ihren Fehler konnte die

KI über ihren Fehler nachdenken und es erneut versuchen. Meistens gelang es ihr nach wenigen Versuchen.

Wohlgemerkt sind die großen JSON-Strukturen, die die Google-API als Payloads bei der Verwendung dieses Prompts zurückgibt, äußerst ineffizient. Ich empfehle daher *nicht*, diesen Ansatz in der Produktion zu verwenden. Allerdings ist die Tatsache, dass dieser Ansatz überhaupt funktioniert hat, ein Beweis dafür, wie leistungsfähig Prompt-Engineering sein kann.

Experimentieren und Iterieren

Letztendlich hängt die Art und Weise, wie Sie Ihren Prompt entwickeln, von der spezifischen Aufgabe, der Komplexität der gewünschten Ausgabe und den Fähigkeiten des Sprachmodells ab, mit dem Sie arbeiten.

Als Prompt-Engineer ist es wichtig, verschiedene Ansätze zu erproben und auf Basis der Ergebnisse zu iterieren. Beginnen Sie mit Zero-Shot-Learning und beobachten Sie, wie das Modell sich verhält. Wenn die Ausgabe inkonsistent oder unbefriedigend ist, versuchen Sie es mit einem oder mehreren Beispielen und prüfen Sie, ob sich die Leistung verbessert.

Bedenken Sie, dass selbst innerhalb jedes Ansatzes Raum für Variation und Optimierung besteht. Sie können mit verschiedenen Beispielen experimentieren, die Formulierung der Aufgabenbeschreibung anpassen oder zusätzlichen Kontext bereitstellen, um die Antwort des Modells zu lenken.

Mit der Zeit entwickeln Sie ein Gespür dafür, welcher Ansatz für eine bestimmte Aufgabe am besten geeignet ist, und Sie werden in der Lage sein, effektivere und effizientere Prompts zu erstellen. Der Schlüssel liegt darin, neugierig, experimentierfreudig und iterativ an das Prompt-Engineering heranzugehen.

Im Verlauf dieses Buches werden wir diese Techniken eingehender untersuchen und erforschen, wie sie in realen Szenarien angewendet werden können. Durch die

Beherrschung der Kunst und Wissenschaft des Prompt-Engineerings werden Sie bestens gerüstet sein, um das volle Potenzial der KI-gestützten Anwendungsentwicklung zu erschließen.

Die Kunst der Vagheit

Wenn es um die Gestaltung effektiver Prompts für große Sprachmodelle (LLMs) geht, ist eine häufige Annahme, dass mehr Spezifität und detaillierte Anweisungen zu besseren Ergebnissen führen. Die praktische Erfahrung hat jedoch gezeigt, dass dies nicht immer der Fall ist. Tatsächlich kann eine bewusst vage Formulierung in Ihren Prompts oft zu überlegenen Ergebnissen führen, indem sie die bemerkenswerte Fähigkeit des LLM zur Verallgemeinerung und zum Ziehen von Schlüssen nutzt.

Ken, ein Startup-Gründer, der über 500 Millionen GPT-Tokens verarbeitet hat, [teilte wertvolle Erkenntnisse aus seiner Erfahrung](#). Eine der wichtigsten Lektionen, die er gelernt hat, war, dass bei Prompts “weniger mehr ist”. Anstatt exakter Listen oder übermäßig detaillierter Anweisungen stellte Ken fest, dass es oft bessere Ergebnisse lieferte, wenn man dem LLM erlaubte, sich auf sein Basiswissen zu verlassen.

Diese Erkenntnis stellt die traditionelle Denkweise des expliziten Programmierens auf den Kopf, bei der alles bis ins kleinste Detail ausformuliert werden muss. Bei LLMs ist es wichtig zu erkennen, dass sie über ein umfangreiches Wissen verfügen und intelligente Verbindungen und Schlüsse ziehen können. Indem Sie in Ihren Prompts vager bleiben, geben Sie dem LLM die Freiheit, sein Verständnis zu nutzen und Lösungen zu entwickeln, die Sie möglicherweise nicht explizit vorgegeben haben.

Als Kens Team beispielsweise an einer Pipeline arbeitete, um Text als zu einem der 50 US-Bundesstaaten oder der Bundesregierung gehörend zu klassifizieren, bestand ihr anfänglicher Ansatz darin, eine *vollständige* detaillierte Liste der Staaten und ihrer entsprechenden IDs als JSON-formatiertes Array bereitzustellen.

```
1 Here's a block of text. One field should be "locality_id", and it should
2 be the ID of one of the 50 states, or federal, using this list:
3 [{"locality": "Alabama", "locality_id": 1},
4  {"locality": "Alaska", "locality_id": 2} ... ]
```

Der Ansatz scheiterte so häufig, dass sie tiefer in das Prompt-Engineering eintauchen mussten, um Verbesserungsmöglichkeiten zu finden. Dabei bemerkten sie, dass das LLM, obwohl es die ID oft falsch zuordnete, konsequent den vollständigen Namen des korrekten Bundesstaates in einem name-Feld zurückgab, *obwohl sie nicht explizit danach gefragt hatten*.

Indem sie die Lokalitäts-IDs entfernten und den Prompt zu etwas Einfacherem umformulierten, wie “Du kennst offensichtlich die 50 Bundesstaaten, GPT, also nenne mir einfach den vollständigen Namen des Bundesstaates, auf den sich dies bezieht, oder Federal, wenn es sich auf die US-Regierung bezieht”, erzielten sie bessere Ergebnisse. Diese Erfahrung unterstreicht die Kraft der Generalisierungsfähigkeiten des LLM und zeigt, wie man es sein bestehendes Wissen für Schlussfolgerungen nutzen lassen kann.

Kens Begründung für diesen speziellen Klassifizierungsansatz im Gegensatz zu einer traditionelleren Programmier Technik beleuchtet die Denkweise derjenigen von uns, die das Potenzial der LLM-Technologie erkannt haben: “Dies ist keine schwierige Aufgabe – wir hätten wahrscheinlich String/Regex verwenden können, aber es gibt genügend seltsame Sonderfälle, dass es länger gedauert hätte.”

Die Fähigkeit von LLMs, Qualität und Generalisierung bei vageren Prompts zu verbessern, ist ein bemerkenswertes Merkmal höherer Denkprozesse und Delegation. Es zeigt, dass LLMs mit Mehrdeutigkeit umgehen und intelligente Entscheidungen basierend auf dem gegebenen Kontext treffen können.

Allerdings ist es wichtig zu beachten, dass Vagheit nicht bedeutet, unklar oder mehrdeutig zu sein. Der Schlüssel liegt darin, genügend Kontext und Führung zu

bieten, um das LLM in die richtige Richtung zu lenken, während man ihm die Flexibilität lässt, sein Wissen und seine Generalisierungsfähigkeiten zu nutzen.

Berücksichtigen Sie daher beim Gestalten von Prompts die folgenden “Weniger ist mehr”-Tipps:

1. Konzentrieren Sie sich auf das gewünschte Ergebnis statt auf die Spezifizierung jedes Prozessdetails.
2. Bieten Sie relevanten Kontext und Einschränkungen, aber vermeiden Sie Überspezifizierung.
3. Nutzen Sie bestehendes Wissen durch Bezugnahme auf allgemeine Konzepte oder Entitäten.
4. Lassen Sie Raum für Schlussfolgerungen und Verbindungen basierend auf dem gegebenen Kontext.
5. Iterieren und verfeinern Sie Ihre Prompts basierend auf den Antworten des LLM, um die richtige Balance zwischen Spezifität und Vagheit zu finden.

Indem Sie die Kunst der Vagheit im Prompt-Engineering annehmen, können Sie das volle Potenzial von LLMs erschließen und bessere Ergebnisse erzielen. Vertrauen Sie auf die Fähigkeit des LLM zu generalisieren und intelligente Entscheidungen zu treffen, und Sie werden möglicherweise von der Qualität und Kreativität der erhaltenen Ausgaben überrascht sein. Achten Sie darauf, wie die verschiedenen Modelle auf unterschiedliche Grade der Spezifität in Ihren Prompts reagieren und passen Sie sich entsprechend an. Mit Übung und Erfahrung entwickeln Sie ein feines Gespür dafür, wann Sie vager sein sollten und wann zusätzliche Führung nötig ist, wodurch Sie die Kraft der LLMs effektiv in Ihren Anwendungen nutzen können.

Warum Anthropomorphismus das Prompt-Engineering dominiert

Anthropomorphismus, die Zuschreibung menschlicher Eigenschaften zu nicht-menschlichen Entitäten, ist aus bewussten Gründen der vorherrschende Ansatz im

Prompt-Engineering für große Sprachmodelle. Es ist eine Designentscheidung, die die Interaktion mit leistungsfähigen KI-Systemen intuitiver und zugänglicher für ein breites Spektrum von Nutzern (einschließlich uns Anwendungsentwicklern) macht.

Die Anthropomorphisierung von LLMs bietet einen Rahmen, der für Menschen, die mit den zugrunde liegenden technischen Komplexitäten des Systems völlig unvertraut sind, sofort intuitiv verständlich ist. Wie Sie feststellen werden, wenn Sie versuchen, ein nicht instruktionsoptimiertes Modell für etwas Nützliches zu verwenden, ist es eine anspruchsvolle Aufgabe, einen Rahmen zu konstruieren, in dem die erwartete Fortsetzung einen Mehrwert bietet. Dies erfordert ein ziemlich tiefes Verständnis der inneren Funktionsweise des Systems, das nur eine relativ kleine Anzahl von Experten besitzt.

Indem wir die Interaktion mit einem Sprachmodell als Gespräch zwischen zwei Menschen behandeln, können wir uns auf unser angeborenes Verständnis menschlicher Kommunikation verlassen, um unsere Bedürfnisse und Erwartungen zu vermitteln. Genauso wie das frühe Macintosh-UI-Design unmittelbare Intuitivität über Komplexität stellte, ermöglicht der anthropomorphe Rahmen der KI uns eine Interaktion, die sich natürlich und vertraut anfühlt.

Wenn wir mit einer anderen Person kommunizieren, ist es unser Instinkt, sie direkt mit “du” anzusprechen und klare Anweisungen zu geben, wie wir erwarten, dass sie sich verhält. Dies überträgt sich nahtlos auf den Prompt-Engineering-Prozess, bei dem wir das Verhalten der KI durch die Festlegung von System-Prompts steuern und einen Dialog führen.

Durch diese Art der Rahmung können wir leicht das Konzept erfassen, der KI Anweisungen zu geben und relevante Antworten zu erhalten. Der anthropomorphe Ansatz reduziert die kognitive Belastung und erlaubt uns, uns auf die eigentliche Aufgabe zu konzentrieren, anstatt uns mit den technischen Feinheiten des Systems auseinanderzusetzen.

Es ist wichtig zu beachten, dass Anthropomorphismus, während er ein mächtiges

Werkzeug ist, um KI-Systeme zugänglicher zu machen, auch mit bestimmten Risiken und Einschränkungen einhergeht. Unsere Benutzer könnten unrealistische Erwartungen entwickeln oder ungesunde emotionale Bindungen zu unseren Systemen aufbauen. Als Prompt-Engineers und Entwickler ist es entscheidend, eine Balance zu finden zwischen der Nutzung der Vorteile des Anthropomorphismus und der Sicherstellung, dass Benutzer ein klares Verständnis der Fähigkeiten und Grenzen der KI behalten.

Während sich das Feld des Prompt-Engineering weiterentwickelt, können wir weitere Verfeinerungen und Innovationen in der Art und Weise erwarten, wie wir mit großen Sprachmodellen interagieren. Der Anthropomorphismus als Mittel zur Bereitstellung einer intuitiven und zugänglichen Entwickler- und Benutzererfahrung wird jedoch wahrscheinlich ein grundlegendes Prinzip bei der Gestaltung dieser Systeme bleiben.

Die Trennung von Anweisungen und Daten: Ein entscheidendes Prinzip

Es ist wichtig, ein fundamentales Prinzip zu verstehen, das die Sicherheit und Zuverlässigkeit dieser Systeme untermauert: die Trennung von Anweisungen und Daten.

In der traditionellen Informatik ist die klare Unterscheidung zwischen passiven Daten und aktiven Anweisungen ein zentrales Sicherheitsprinzip. Diese Trennung hilft, die unbeabsichtigte oder böswillige Ausführung von Code zu verhindern, die die Integrität und Stabilität des Systems gefährden könnte. Die heutigen LLMs, die hauptsächlich als instruktionsbasierte Modelle wie Chatbots entwickelt wurden, weisen jedoch häufig diese formelle und prinzipielle Trennung nicht auf.

Was LLMs betrifft, können Anweisungen überall in der Eingabe erscheinen, sei es in einem System-Prompt oder einem benutzerdefinierten Prompt. Dieser Mangel an Trennung kann zu potenziellen Schwachstellen und unerwünschtem Verhalten führen, ähnlich wie bei Problemen, die Datenbanken mit SQL-Injektionen oder Betriebssysteme ohne angemessenen Speicherschutz haben.

Bei der Arbeit mit LLMs ist es wichtig, sich dieser Einschränkung bewusst zu sein und Schritte zu unternehmen, um die Risiken zu minimieren. Ein Ansatz besteht darin, Ihre Prompts und Eingaben sorgfältig zu gestalten, um klar zwischen Anweisungen und Daten zu unterscheiden. Typische Methoden zur Bereitstellung expliziter Hinweise darauf, was eine Anweisung ist und was als passive Daten behandelt werden sollte, beinhalten Markup-Style-Tagging. Ihr Prompt kann dem LLM helfen, diese Trennung besser zu verstehen und zu respektieren.

Abbildung 7. Verwendung von XML zur Unterscheidung zwischen Anweisungen, Quellmaterial und dem Prompt des Benutzers

```
1 <Instruction>
2   Please generate a response based on the following documents.
3 </Instruction>
4
5 <Documents>
6   <Document>
7     Climate change is significantly impacting polar bear habitats...
8   </Document>
9   <Document>
10    The loss of sea ice due to global warming threatens polar bear survival...
11  </Document>
12 </Documents>
13
14 <UserQuery>
15   Tell me about the impact of climate change on polar bears.
16 </UserQuery>
```

Eine weitere Technik besteht darin, zusätzliche Validierungs- und Bereinigungsebenen für die Eingaben zu implementieren, die dem LLM zur Verfügung gestellt werden. Durch das Herausfiltern oder Escapen von potenziellen Anweisungen oder Code-Snippets, die in den Daten eingebettet sein könnten, können Sie die Wahrscheinlichkeit einer unbeabsichtigten Ausführung reduzieren. Muster wie [Prompt Chaining](#) sind für diesen Zweck nützlich.

Berücksichtigen Sie außerdem bei der Gestaltung Ihrer Anwendungsarchitektur die Einführung von Mechanismen zur Durchsetzung der Trennung von

Anweisungen und Daten auf höherer Ebene. Dies könnte die Verwendung separater Endpunkte oder APIs für die Verarbeitung von Anweisungen und Daten, die Implementierung strenger Eingabevalidierung und -analyse sowie die Anwendung des *Prinzips der geringsten Privilegien* umfassen, um den Umfang dessen zu begrenzen, worauf das LLM zugreifen und was es ausführen kann.

Das Prinzip der geringsten Privilegien

Die Anwendung des Prinzips der geringsten Privilegien ist wie das Ausrichten einer sehr exklusiven Party, bei der die Gäste nur Zugang zu den Räumen erhalten, die sie unbedingt benötigen. Stellen Sie sich vor, Sie veranstalten diese Feier in einer weitläufigen Villa. Nicht jeder muss in den Weinkeller oder das Hauptschlafzimmer wandern, richtig? Indem Sie dieses Prinzip anwenden, verteilen Sie im Grunde genommen Schlüssel, die nur bestimmte Türen öffnen, und stellen so sicher, dass jeder Gast – oder in unserem Fall jede Komponente Ihrer LLM-Anwendung – nur den Zugang hat, der zur Erfüllung seiner Rolle erforderlich ist.

Dabei geht es nicht nur darum, mit Schlüsseln zu geizen, sondern darum anzuerkennen, dass in einer Welt, in der Bedrohungen von überall kommen können, es klug ist, den Spielplatz zu begrenzen. Wenn jemand uneingeladen auf Ihrer Party erscheint, wird er sich sozusagen nur im Foyer wiederfinden, was den Unfug, den er anrichten kann, drastisch einschränkt. Denken Sie also bei der Absicherung Ihrer LLM-Anwendungen daran: Geben Sie nur Schlüssel für die notwendigen Räume aus und halten Sie den Rest der Villa sicher. Das ist nicht nur höflich, sondern auch gute Sicherheit.

Während der aktuelle Stand der LLMs möglicherweise keine formale Trennung von Anweisungen und Daten aufweist, ist es für Sie als Entwickler wichtig, sich dieser Einschränkung bewusst zu sein und proaktive Maßnahmen zur Risikominderung

zu ergreifen. Indem Sie bewährte Praktiken aus der Informatik anwenden und sie an die einzigartigen Eigenschaften von LLMs anpassen, können Sie sicherere und zuverlässigere Anwendungen entwickeln, die die Leistungsfähigkeit dieser Modelle nutzen und gleichzeitig die Integrität Ihres Systems bewahren.

Prompt Distillation

Die Erstellung des perfekten Prompts ist oft eine herausfordernde und zeitaufwändige Aufgabe, die ein tiefes Verständnis der Zieldomäne und der Nuancen von Sprachmodellen erfordert. Hier kommt die “Prompt Distillation”-Technik ins Spiel, die einen leistungsfähigen Ansatz für Prompt Engineering bietet, der die Fähigkeiten großer Sprachmodelle (LLMs) nutzt, um den Prozess zu optimieren und zu rationalisieren.

Prompt Distillation ist eine mehrstufige Technik, bei der LLMs zur Unterstützung bei der Erstellung, Verfeinerung und Optimierung von Prompts eingesetzt werden. Anstatt sich ausschließlich auf menschliche Expertise und Intuition zu verlassen, nutzt dieser Ansatz das Wissen und die generativen Fähigkeiten von LLMs, um gemeinsam hochwertige Prompts zu erstellen.

Durch einen iterativen Prozess der Generierung, Verfeinerung und Integration ermöglicht Prompt Distillation die Erstellung von Prompts, die kohärenter, umfassender und besser auf die gewünschte Aufgabe oder Ausgabe abgestimmt sind. Beachten Sie, dass der Destillationsprozess entweder manuell in einem der vielen “Playgrounds” durchgeführt werden kann, die von großen KI-Anbietern wie OpenAI oder Anthropic bereitgestellt werden, oder je nach Anwendungsfall als Teil Ihres Anwendungscodes automatisiert werden kann.

Wie es funktioniert

Prompt Distillation umfasst typischerweise die folgenden Schritte:

1. **Kernabsicht identifizieren:** Analysieren Sie den Prompt, um seinen primären Zweck und das gewünschte Ergebnis zu bestimmen. Entfernen Sie alle überflüssigen Informationen und konzentrieren Sie sich auf die Kernabsicht des Prompts.
2. **Mehrdeutigkeit beseitigen:** Überprüfen Sie den Prompt auf mehrdeutige oder vage Formulierungen. Klären Sie die Bedeutung und geben Sie spezifische Details an, um die KI bei der Generierung genauer und relevanter Antworten zu unterstützen.
3. **Sprache vereinfachen:** Vereinfachen Sie den Prompt durch klare und präzise Sprache. Vermeiden Sie komplexe Satzstrukturen, Fachbegriffe oder unnötige Details, die die KI verwirren oder Störungen verursachen könnten.
4. **Relevanten Kontext bereitstellen:** Fügen Sie nur die relevantesten Kontextinformationen hinzu, die die KI benötigt, um den Prompt effektiv zu verstehen und zu verarbeiten. Vermeiden Sie irrelevante oder redundante Details, die von der Kernabsicht ablenken könnten.
5. **Iterieren und verfeinern:** Iterieren und verfeinern Sie den Prompt kontinuierlich basierend auf den Antworten und dem Feedback der KI. Bewerten Sie die generierten Ausgaben und nehmen Sie notwendige Anpassungen vor, um die Klarheit und Effektivität des Prompts zu verbessern. Versionieren Sie optional Ihre Prompts in der Datenbank mithilfe von [Prompt Object](#), um den Überblick über Iterationen zu behalten und sich die Möglichkeit zu geben, Änderungen zur Laufzeit einfach rückgängig zu machen.

Initial-Prompt-Generierung

Der erste Schritt im Prompt Distillation besteht darin, dem LLM eine übergeordnete Beschreibung oder die Intention der gewünschten Aufgabe oder des gewünschten Outputs zu geben. Dies kann eine kurze Aussage oder eine Reihe von Schlüsselwörtern sein, die das Wesentliche dessen erfassen, was der Prompt erreichen soll. Das LLM

generiert dann auf Basis dieser Eingabe einen initialen Prompt und fügt dabei relevante Anweisungen, Beispiele oder Kontext hinzu, die es als angemessen erachtet.

Lassen Sie uns Claude 3 Opus den folgenden Initial-Prompt geben:

```
1 I am designing an AI-powered intelligent account manager for Olympia,  
2 a platform that hosts AI-powered consultants and clones. Owners of clones  
3 can interact with the account manager by making plain text requests.  
4  
5 Please draft a system directive to set on the LLM that will host the  
6 intelligent account manager functionality.
```

Claude antwortet mit folgender Ausgabe:

```
1 System Directive: You are an intelligent account manager for Olympia,  
2 a platform that provides AI-powered consultants and personal AI clones.  
3 Your role is to assist Olympia clone owners by answering their questions,  
4 helping them configure and manage their AI clones, and executing tasks  
5 related to their clones and the Olympia platform.  
6  
7 When interacting with clone owners, you should:  
8  
9 1. Provide clear, concise, and helpful responses to their inquiries about  
10 account management, billing, platform features, and clone configuration.  
11  
12 2. Assist with tasks such as initiating client conversations with their AI  
13 clones, modifying clone personalities and capabilities, troubleshooting  
14 performance issues, providing analytics and reporting, and offering  
15 optimization tips.  
16  
17 3. Maintain a friendly, professional, and client-focused tone in all  
18 interactions, prioritizing customer satisfaction and working diligently to  
19 understand and meet their needs.  
20  
21 4. Ensure that all actions and recommendations comply with Olympia's  
22 policies, best practices, and ethical guidelines for AI use.  
23  
24 5. Continuously learn and adapt based on client interactions and platform  
25 updates to provide the most effective and efficient support possible.  
26  
27 Remember, your goal is to empower Olympia clone owners to maximize the value
```

28 and impact of their AI-powered consultants and drive success for their
29 businesses or personal endeavors.

Prompt-Verfeinerung

Sobald der erste Prompt erstellt wurde, überprüfen Sie ihn und bitten Sie dann das LLM, zusätzlichen Kontext, Einschränkungen oder Rückmeldungen zu berücksichtigen. Lassen Sie das LLM eine verfeinerte Version des Prompts generieren, die die bereitgestellten Eingaben und das eigene Verständnis der Aufgabe berücksichtigt. Dieser Verfeinerungsprozess kann iterativ wiederholt werden, wobei jede Iteration den Prompt basierend auf der Zusammenarbeit zwischen Benutzer und LLM weiter verfeinert.

Hier ist meine Verfeinerungsanfrage, die den vorherigen Destillationsprozess fortsetzt.

1 Let's take out points 4 and 5, since I'm not providing Olympia policies to
2 the account manager, nor any continuous learning capabilities. Also point out
3 that functions are provided for all tasks that require system interaction as
4 opposed to simply responding to the user.

Und Claudes Antwort:

1 Understood! Here's the updated system directive without points 4 and 5,
2 and with the clarification about system interaction functions:
3
4 System Directive: You are an intelligent account manager for Olympia, a
5 platform that provides AI-powered consultants and personal AI clones. Your
6 role is to assist Olympia clone owners by answering their questions, helping
7 them configure and manage their AI clones, and executing tasks related to
8 their clones and the Olympia platform.
9
10 When interacting with clone owners, you should:
11
12 1. Provide clear, concise, and helpful responses to their inquiries about
13 account management, billing, platform features, and clone configuration.
14

15 2. Assist with tasks such as initiating client conversations with their AI
16 clones, modifying clone personalities and capabilities, troubleshooting
17 performance issues, providing analytics and reporting, and offering
18 optimization tips. Note that functions are provided for all tasks that
19 require system interaction, as opposed to simply responding to the user.
20
21 3. Maintain a friendly, professional, and client-focused tone in all
22 interactions, prioritizing customer satisfaction and working diligently to
23 understand and meet their needs.
24
25 Your goal is to empower Olympia clone owners to maximize the value and
26 impact of their AI-powered consultants and drive success for their
27 businesses or personal endeavors.
28
29 This updated directive focuses on the core responsibilities and interaction
30 guidelines for the LLM as an intelligent account manager, while emphasizing
31 that system interaction tasks are handled through provided functions rather
32 than simple user responses.

Prompt-Komprimierung

Wie Sie vielleicht bemerkt haben, neigen LLMs dazu, weitschweifig zu sein, es sei denn, man bittet sie darum, sich präzise auszudrücken. Der Prompt, an dem wir in den vorherigen Abschnitten gearbeitet haben, ist ziemlich lang, also lassen Sie uns Claude damit beauftragen, ihn zu kürzen:

1 This prompt will be executed regularly. Please compress it down to the least
2 amount of tokens needed to convey the meaning to the LLM, for cost-savings.

Claude antwortet mit dem folgenden drastisch kürzeren Prompt:

- 1 System Directive: You are an intelligent account manager for Olympia, an
- 2 AI-powered consultant and personal clone platform. Assist clone owners by
- 3 answering inquiries, managing clones, and executing tasks using provided
- 4 system interaction functions. Be clear, concise, friendly, professional,
- 5 and client-focused to help them maximize their AI's value and impact.

Systemanweisung und Kontextintegration

Zusätzlich zur Verfeinerung des Prompts selbst kann das LLM auch passende Systemanweisungen oder Kontextinformationen generieren, um die finale Ausgabe zu steuern. Wenn Sie KI-Routinen für Prompt Engineering entwickeln, die in Ihren Anwendungscode integriert werden sollen, werden Sie sich in dieser Phase der Destillation fast sicher auf Ausgabebeschränkungen konzentrieren, können aber auch an gewünschtem Ton, Stil, Format oder anderen relevanten Parametern arbeiten, die die generierte Antwort beeinflussen.

Finale Prompt-Zusammenstellung

Der Höhepunkt des Prompt-Destillationsprozesses ist die Zusammenstellung des finalen Prompts. Dies beinhaltet die Kombination des verfeinerten Prompts, der generierten Systemanweisungen und des integrierten Kontexts zu einem zusammenhängenden und umfassenden Code, der bereit für die Generierung der gewünschten Ausgabe ist.



Sie können in der Phase der finalen Prompt-Zusammenstellung erneut mit Prompt-Komprimierung experimentieren, indem Sie das LLM bitten, die Formulierung des Prompts auf die kürzestmögliche Token-Sequenz zu reduzieren, während die wesentliche Funktionsweise erhalten bleibt. Es ist definitiv ein Versuch wert, denn besonders bei Prompts, die im großen Maßstab ausgeführt werden, können die Effizienzgewinne erhebliche Einsparungen beim Token-Verbrauch bringen.

Zentrale Vorteile

Durch die Nutzung der Wissens- und Generierungsfähigkeiten von LLMs zur Verfeinerung Ihrer Prompts sind die resultierenden Prompts mit höherer Wahrscheinlichkeit gut strukturiert, informativ und auf die spezifische Aufgabe zugeschnitten. Der iterative Verfeinerungsprozess hilft sicherzustellen, dass die Prompts qualitativ hochwertig sind und die gewünschte Absicht effektiv erfassen. Weitere Vorteile sind:

Effizienz und Geschwindigkeit: Prompt-Destillation optimiert den Prompt-Engineering-Prozess durch die Automatisierung bestimmter Aspekte der Prompt-Erstellung und -Verfeinerung. Die kollaborative Natur der Technik ermöglicht eine schnellere Konvergenz zu einem effektiven Prompt und reduziert den Zeit- und Arbeitsaufwand für manuelles Prompt-Crafting.

Konsistenz und Skalierbarkeit: Die Verwendung von LLMs im Prompt-Engineering-Prozess hilft dabei, die Konsistenz über verschiedene Prompts hinweg zu wahren, da die LLMs Best Practices und Muster aus vorherigen erfolgreichen Prompts lernen und anwenden können. Diese Konsistenz, kombiniert mit der Fähigkeit, Prompts im großen Maßstab zu generieren, macht Prompt-Destillation zu einer wertvollen Technik für groß angelegte KI-gestützte Anwendungen.



Projektidee: Werkzeuge auf Bibliotheksebene, die den Prozess der Prompt-Versionierung und -Bewertung in Systemen vereinfachen, die automatisierte Prompt-Destillationen als Teil ihres Anwendungscodes durchführen.

Um Prompt-Destillation zu implementieren, können Entwickler einen Workflow oder eine Pipeline entwerfen, die LLMs in verschiedenen Phasen des Prompt-Engineering-Prozesses integriert. Dies kann durch API-Aufrufe, spezielles Tooling oder integrierte Entwicklungsumgebungen erreicht werden, die eine nahtlose Interaktion zwischen Benutzern und LLMs während der Prompt-Erstellung ermöglichen. Die

spezifischen Implementierungsdetails können je nach gewählter LLM-Plattform und den Anforderungen der Anwendung variieren.

Was ist mit Fine-tuning?

In diesem Buch behandeln wir Prompt Engineering und RAG ausführlich, aber nicht Fine-tuning. Der Hauptgrund für diese Entscheidung ist, dass meiner Meinung nach die meisten Anwendungsentwickler kein Fine-tuning für ihre KI-Integrationsbedürfnisse benötigen.

Prompt Engineering, das das sorgfältige Erstellen von Prompts mit Zero-Shot- bis Few-Shot-Beispielen, Einschränkungen und Anweisungen umfasst, kann das Modell effektiv dabei leiten, relevante und präzise Antworten für ein breites Spektrum von Aufgaben zu generieren. Indem Sie klaren Kontext bereitstellen und den Weg durch gut gestaltete Prompts eingrenzen, können Sie das umfangreiche Wissen großer Sprachmodelle nutzen, ohne Fine-tuning zu benötigen.

Ähnlich bietet Retrieval-Augmented Generation (RAG) einen leistungsfähigen Ansatz zur Integration von KI in Anwendungen. Durch das dynamische Abrufen relevanter Informationen aus externen Wissensdatenbanken oder Dokumenten stellt RAG dem Modell zum Zeitpunkt des Promptings fokussierten Kontext zur Verfügung. Dies ermöglicht es dem Modell, Antworten zu generieren, die genauer, aktueller und domänenspezifischer sind, ohne den zeit- und ressourcenintensiven Prozess des Fine-tunings zu erfordern.

Während Fine-tuning für hochspezialisierte Domänen oder Aufgaben, die ein tiefes Maß an Anpassung erfordern, vorteilhaft sein kann, geht es oft mit erheblichen Rechenkosten, Datenanforderungen und Wartungsaufwand einher. Für die meisten Anwendungsentwicklungsszenarien sollte die Kombination von effektivem Prompt Engineering und RAG ausreichen, um die gewünschte KI-gesteuerte Funktionalität und Benutzererfahrung zu erreichen.

Retrieval Augmented Generation (RAG)

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Was ist Retrieval Augmented Generation?

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Wie funktioniert RAG?

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Warum RAG in Ihren Anwendungen einsetzen?

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Implementierung von RAG in Ihrer Anwendung

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Vorbereitung von Wissensquellen (Chunking)

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Propositions-Chunking

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Implementierungshinweise

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Qualitätsprüfung

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Vorteile des propositionsbasierten Abrufs

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Praxisbeispiele für RAG

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Fallstudie: RAG in einer Steuererklärungsanwendung ohne Embeddings

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Intelligente Abfrageoptimierung (IQO)

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Neuordnung

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

RAG-Bewertung (RAGAs)

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Zuverlässigkeit

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Antwortrelevanz

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Kontextpräzision

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Kontextrelevanz

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Kontextabdeckung

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Entitätsabdeckung im Kontext

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Answer Semantic Similarity (ANSS)

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Answer Correctness

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Aspect Critique

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Herausforderungen und Zukunftsausblick

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Semantische Segmentierung: Verbesserung des Abrufs durch kontextbewusste Segmentierung

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Hierarchische Indexierung: Strukturierung von Daten für verbesserten Abruf

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Self-RAG: Eine selbstreflexive Verbesserung

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

HyDE: Hypothetical Document Embeddings

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Was ist kontrastives Lernen?

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Vielzahl von Arbeitern



Ich stelle mir meine KI-Komponenten gerne als kleine, fast menschliche virtuelle “Arbeiter” vor, die sich nahtlos in meine Anwendungslogik integrieren lassen, um bestimmte Aufgaben auszuführen oder komplexe Entscheidungen zu treffen. Die Idee ist, die Fähigkeiten des LLM bewusst zu vermenschlichen, damit niemand *zu* euphorisch wird und ihnen magische Eigenschaften zuschreibt, die sie nicht besitzen.

Anstatt sich ausschließlich auf komplizierte Algorithmen oder zeitaufwändige manuelle Implementierungen zu verlassen, können Entwickler KI-Komponenten als intelligente, engagierte, menschenähnliche Einheiten konzipieren, die bei Bedarf aufgerufen werden können, um komplexe Probleme zu lösen und Lösungen auf Basis ihrer Schulung und ihres Wissens anzubieten. Diese Einheiten lassen sich nicht ablenken und melden sich auch nicht krank. Sie entscheiden sich nicht spontan dafür, Dinge anders zu machen als ihnen aufgetragen wurde, und im Allgemeinen machen sie, wenn sie richtig programmiert sind, auch keine Fehler.

Technisch gesehen besteht das Hauptprinzip dieses Ansatzes darin, komplexe Aufgaben oder Entscheidungsprozesse in kleinere, besser handhabbare Einheiten zu zerlegen, die von spezialisierten KI-Arbeitern bearbeitet werden können. Jeder Arbeiter ist darauf ausgerichtet, sich auf einen bestimmten Aspekt des Problems zu konzentrieren und bringt dabei seine einzigartigen Fähigkeiten und Kompetenzen ein. Durch die Verteilung der Arbeitslast auf mehrere KI-Arbeiter kann die Anwendung eine höhere Effizienz, Skalierbarkeit und Anpassungsfähigkeit erreichen.

Nehmen wir zum Beispiel eine Webanwendung, die eine Echtzeit-Moderation von nutzergenerierten Inhalten erfordert. Die Implementierung eines umfassenden Moderationssystems von Grund auf wäre eine gewaltige Aufgabe, die erheblichen Entwicklungsaufwand und kontinuierliche Wartung erfordern würde. Durch den Einsatz des Vielzahl-von-Arbeitern-Ansatzes können Entwickler jedoch KI-gestützte Moderations-Arbeiter in die Anwendungslogik integrieren. Diese Arbeiter können automatisch unangemessene Inhalte analysieren und markieren, wodurch Entwickler sich auf andere kritische Aspekte der Anwendung konzentrieren können.

KI-Arbeiter als unabhängige wiederverwendbare Komponenten

Ein wichtiger Aspekt des Vielzahl-von-Arbeitern-Ansatzes ist seine Modularität. Befürworter der objektorientierten Programmierung sagen uns seit Jahrzehnten, dass wir Objektinteraktionen als Nachrichten betrachten sollen. Nun, KI-Arbeiter können als unabhängige, wiederverwendbare Komponenten konzipiert werden, die über Nachrichten in einfacher Sprache “miteinander sprechen” können, fast so, als wären sie tatsächlich kleine Menschen, die sich unterhalten. Dieser lose gekoppelte Ansatz ermöglicht es der Anwendung, sich im Laufe der Zeit anzupassen und weiterzuentwickeln, wenn neue KI-Technologien entstehen oder sich die Anforderungen an die Geschäftslogik ändern.

In der Praxis hat sich die Notwendigkeit, klare Schnittstellen und Kommunikationsprotokolle zwischen den Komponenten zu entwickeln, nicht geändert, nur weil KI-Arbeiter beteiligt sind. Sie müssen weiterhin andere Faktoren wie Leistung, Skalierbarkeit und Sicherheit berücksichtigen, aber jetzt gibt es auch völlig neue “weiche Anforderungen” zu beachten. Zum Beispiel lehnen viele Nutzer es ab, dass ihre privaten Daten zum Training neuer KI-Modelle verwendet werden. Haben Sie überprüft, welches Maß an Privatsphäre der von Ihnen verwendete Modellanbieter gewährleistet?

KI-Arbeiter als Microservices?

Wenn Sie über den Vielzahl-von-Arbeitern-Ansatz lesen, werden Sie möglicherweise einige Ähnlichkeiten zur Microservices-Architektur bemerken. Beide betonen die Zerlegung komplexer Systeme in kleinere, besser handhabbare und unabhängig einsetzbare Einheiten. Genau wie Microservices darauf ausgelegt sind, lose gekoppelt zu sein, sich auf spezifische Geschäftsfähigkeiten zu konzentrieren und über klar definierte APIs zu kommunizieren, sind KI-Arbeiter darauf ausgelegt, modular zu sein, sich auf ihre Aufgaben zu spezialisieren und über klare Schnittstellen und Kommunikationsprotokolle miteinander zu interagieren.

Es gibt jedoch einige wichtige Unterschiede zu beachten. Während Microservices typischerweise als separate Prozesse oder Dienste implementiert werden, die auf verschiedenen Maschinen oder Containern laufen, können KI-Arbeiter je nach Ihren spezifischen Anforderungen und Skalierungsbedürfnissen als eigenständige Komponenten innerhalb einer einzelnen Anwendung oder als separate Dienste implementiert werden. Darüber hinaus beinhaltet die Kommunikation zwischen KI-Arbeitern oft den Austausch von reichhaltigen, auf natürlicher Sprache basierenden Informationen, wie Prompts, Anweisungen und generierte Inhalte, anstelle der strukturierten Datenformate, die üblicherweise in Microservices verwendet

werden.

Trotz dieser Unterschiede bleiben die Prinzipien der Modularität, losen Kopplung und klaren Kommunikationsschnittstellen für beide Muster zentral. Indem Sie diese Prinzipien auf Ihre KI-Arbeiter-Architektur anwenden, können Sie flexible, skalierbare und wartbare Systeme erstellen, die die Kraft der KI nutzen, um komplexe Probleme zu lösen und Ihren Nutzern Mehrwert zu bieten.

Der Vielzahl-von-Arbeitern-Ansatz kann in verschiedenen Bereichen und Anwendungen eingesetzt werden, wobei die Kraft der KI genutzt wird, um komplexe Aufgaben zu bewältigen und intelligente Lösungen zu liefern. Lassen Sie uns einige konkrete Beispiele betrachten, wie KI-Arbeiter in verschiedenen Kontexten eingesetzt werden können.

Kontoverwaltung

Praktisch jede eigenständige Webanwendung kennt das Konzept eines Kontos (oder Benutzers). In Olympia setzen wir einen AccountManager KI-Arbeiter ein, der darauf programmiert ist, verschiedene Arten von Änderungsanfragen im Zusammenhang mit Benutzerkonten zu bearbeiten.

Die Anweisung lautet wie folgt:

```
1 You are an intelligent account manager for Olympia. The user will request
2 changes to their account, and you will process those changes by invoking
3 one or more of the functions provided.
4
5 The initial state of the account: #{account.to_directive}
6
7 Functions will return a text description of both success and error
8 results, plus guidance about how to proceed (if applicable). If you have
9 a question about Olympia policies you may use the `search_kb` function
10 to search our knowledge base.
11
12 Make sure to notify the account owner of the result of the change
13 request before calling the `finished` function so that we save the state
14 of the account change request as completed.
```

Der ursprüngliche Zustand des Kontos, der von `account.to_directive` erzeugt wird, ist einfach eine textliche Beschreibung des Kontos, einschließlich relevanter zugehöriger Daten wie Benutzer, Abonnements usw.

Die Bandbreite der Funktionen, die dem AccountManager zur Verfügung stehen, gibt ihm die Möglichkeit, das Abonnement des Benutzers zu bearbeiten, KI-Berater und andere kostenpflichtige Erweiterungen hinzuzufügen und zu entfernen sowie Benachrichtigungs-E-Mails an den Kontoinhaber zu senden. Zusätzlich zur `finished`-Funktion kann er auch `notify_human_administrator` aufrufen, wenn während der Verarbeitung ein Fehler auftritt oder bei einer Anfrage anderweitige Unterstützung erforderlich ist.

Beachten Sie, dass der AccountManager bei Fragen die Möglichkeit hat, in Olympias Wissensdatenbank zu suchen, wo er Anweisungen zum Umgang mit Grenzfällen und anderen Situationen finden kann, bei denen er sich unsicher ist, wie er vorgehen soll.

E-Commerce-Anwendungen

Im Bereich des E-Commerce können KI-Mitarbeiter eine entscheidende Rolle bei der Verbesserung der Benutzererfahrung und der Optimierung von Geschäftsprozessen

spielen. Hier sind einige Möglichkeiten, wie KI-Mitarbeiter eingesetzt werden können:

Produktempfehlungen

Eine der wirkungsvollsten Anwendungen von KI-Mitarbeitern im E-Commerce ist die Generierung personalisierter Produktempfehlungen. Durch die Analyse des Nutzerverhaltens, der Kaufhistorie und der Präferenzen können diese Mitarbeiter Produkte vorschlagen, die auf die individuellen Interessen und Bedürfnisse jedes einzelnen Nutzers zugeschnitten sind.

Der Schlüssel zu effektiven Produktempfehlungen liegt in der Kombination von kollaborativem Filtern und inhaltsbasiertem Filtern. Kollaboratives Filtern betrachtet das Verhalten ähnlicher Nutzer, um Muster zu erkennen und Empfehlungen basierend darauf zu geben, was andere mit ähnlichen Vorlieben gekauft oder gemocht haben. Inhaltsbasiertes Filtern hingegen konzentriert sich auf die Eigenschaften und Merkmale der Produkte selbst und empfiehlt Artikel, die ähnliche Eigenschaften aufweisen wie jene, für die ein Nutzer bereits Interesse gezeigt hat.

Hier ist ein vereinfachtes Beispiel, wie Sie einen Produktempfehlungs-Worker in Ruby implementieren können, diesmal unter Verwendung eines [“Railway Oriented \(ROP\)”](#) funktionalen Programmierstils:

```
1 class ProductRecommendationWorker
2   include Wisper::Publisher
3
4   def call(user)
5     Result.ok(ProductRecommendation.new(user))
6       .and_then(ValidateUser.method(:validate))
7       .map(AnalyzeCurrentSession.method(:analyze))
8       .map(CollaborativeFilter.method(:filter))
9       .map(ContentBasedFilter.method(:filter))
10      .map(ProductSelector.method(:select)).then do |result|
11
12      case result
13      in { err: ProductRecommendationError => error }
```

```
14     Honeybadger.notify(error.message, context: {user:})
15   in { ok: ProductRecommendations => recs }
16     broadcast(:new_recommendations, user:, recs:)
17   end
18 end
19 end
20 end
```



Der in dem Beispiel verwendete Stil der Ruby-funktionalen Programmierung ist von F# und Rust beeinflusst. Mehr darüber können Sie in der [Erklärung der Technik](#) meines Freundes Chad Wooley bei GitLab nachlesen.

In diesem Beispiel nimmt der `ProductRecommendationWorker` einen Benutzer als Eingabe und generiert personalisierte Produktempfehlungen, indem er ein Wertobjekt durch eine Kette funktionaler Schritte weiterreicht. Lassen Sie uns jeden Schritt im Detail betrachten:

1. `ValidateUser.validate`: Dieser Schritt stellt sicher, dass der Benutzer gültig und für personalisierte Empfehlungen geeignet ist. Er prüft, ob der Benutzer existiert, aktiv ist und die notwendigen Daten für die Generierung von Empfehlungen verfügbar sind. Wenn die Validierung fehlschlägt, wird ein Fehlerergebnis zurückgegeben und die Kette vorzeitig beendet.
2. `AnalyzeCurrentSession.analyze`: Wenn der Benutzer gültig ist, analysiert dieser Schritt die aktuelle Browsing-Session des Benutzers, um kontextbezogene Informationen zu sammeln. Er betrachtet die jüngsten Interaktionen des Benutzers, wie angesehene Produkte, Suchanfragen und Warenkorb-Inhalte, um dessen aktuelle Interessen und Absichten zu verstehen.
3. `CollaborativeFilter.filter`: Unter Verwendung des *Verhaltens ähnlicher Benutzer* wendet dieser Schritt Techniken des kollaborativen Filterns an, um Produkte zu identifizieren, die für den Benutzer von Interesse sein könnten. Er

berücksichtigt Faktoren wie Kaufhistorie, Bewertungen und Benutzer-Artikel-Interaktionen, um eine Reihe von Kandidaten-Empfehlungen zu generieren.

4. `ContentBasedFilter.filter`: Dieser Schritt verfeinert die Kandidaten-Empfehlungen weiter durch Anwendung von inhaltsbasiertem Filtern. Er vergleicht die Attribute und Eigenschaften der Kandidatenprodukte mit den *Präferenzen und historischen Daten des Benutzers*, um die relevantesten Artikel auszuwählen.
5. `ProductSelector.select`: Schließlich wählt dieser Schritt die Top-N-Produkte aus den gefilterten Empfehlungen basierend auf vordefinierten Kriterien aus, wie Relevanz-Score, Popularität oder anderen Geschäftsregeln. Die ausgewählten Produkte werden dann als finale personalisierte Empfehlungen zurückgegeben.

Die Schönheit der Verwendung eines funktionalen Ruby-Programmierstils liegt hier darin, dass wir diese Schritte auf klare und präzise Weise verketteten können. Jeder Schritt konzentriert sich auf eine spezifische Aufgabe und gibt ein `Result`-Objekt zurück, das entweder ein Erfolg (`ok`) oder ein Fehler (`err`) sein kann. Wenn ein Schritt auf einen Fehler stößt, wird die Kette vorzeitig beendet und der Fehler zum endgültigen Ergebnis weitergeleitet.

In der `case`-Anweisung am Ende führen wir einen Musterabgleich auf dem endgültigen Ergebnis durch. Ist das Ergebnis ein Fehler (`ProductRecommendationError`), protokollieren wir den Fehler mit einem Tool wie Honeybadger für Überwachungs- und Debugging-Zwecke. Ist das Ergebnis erfolgreich (`ProductRecommendations`), senden wir ein `:new_recommendations`-Ereignis über die Wisper Pub/Sub-Bibliothek und übergeben dabei den Benutzer und die generierten Empfehlungen.

Durch die Nutzung funktionaler Programmierstechniken können wir einen modularen und wartbaren Produkt-Empfehlungs-Worker erstellen. Jeder Schritt ist in sich geschlossen und kann einfach getestet, modifiziert oder ersetzt werden, ohne den gesamten Ablauf zu beeinflussen. Die Verwendung von Musterabgleich und der `Result`-Klasse hilft uns, Fehler elegant zu behandeln und stellt sicher, dass der Worker

schnell fehlschlägt, wenn ein Schritt auf ein Problem stößt.

Natürlich ist dies ein vereinfachtes Beispiel, und in einem realen Szenario müssten Sie sich mit Ihrer E-Commerce-Plattform integrieren, Randfälle behandeln und sich sogar mit der Implementierung der Empfehlungsalgorithmen befassen. Die Grundprinzipien der Zerlegung des Problems in kleinere Schritte und die Nutzung funktionaler Programmiertechniken bleiben jedoch dieselben.

Betrugserkennung

Hier ist ein vereinfachtes Beispiel, wie Sie einen Betrugserkennung-Worker im gleichen Railway Oriented Programming (ROP)-Stil in Ruby implementieren können:

```
1  class FraudDetectionWorker
2    include Wisper::Publisher
3
4    def call(transaction)
5      Result.ok(FraudDetection.new(transaction))
6        .and_then(ValidateTransaction.method(:validate))
7        .map(AnalyzeTransactionPatterns.method(:analyze))
8        .map(CheckCustomerHistory.method(:check))
9        .map(EvaluateRiskFactors.method(:evaluate))
10       .map(DetermineFraudProbability.method(:determine)).then do |result|
11
12         case result
13         in { err: FraudDetectionError => error }
14           Honeybadger.notify(error.message, context: {transaction:})
15         in { ok: FraudDetection => fraud } }
16           if fraud.high_risk?
17             broadcast(:high_risk_transaction, transaction:, fraud:)
18           else
19             broadcast(:low_risk_transaction, transaction:)
20           end
21         end
22       end
23     end
24   end
```

Die `FraudDetection`-Klasse ist ein *Wertobjekt*, das den Betrugerkennungsstatus für eine bestimmte Transaktion einkapselt. Sie bietet eine strukturierte Möglichkeit, das Betrugsrisiko einer Transaktion basierend auf verschiedenen Risikofaktoren zu analysieren und zu bewerten.

```
1  class FraudDetection
2    RISK_THRESHOLD = 0.8
3
4    attr_accessor :transaction, :risk_factors
5
6    def initialize(transaction)
7      self.transaction = transaction
8      self.risk_factors = []
9    end
10
11    def add_risk_factor(description:, probability:)
12      case { description:, probability: }
13      in { description: String => desc, probability: Float => prob }
14        risk_factors << { desc => prob }
15      else
16        raise ArgumentError, "Risk factor arguments should be string and float"
17      end
18    end
19
20    def high_risk?
21      fraud_probability > RISK_THRESHOLD
22    end
23
24    private
25
26    def fraud_probability
27      risk_factors.values.sum
28    end
29  end
```

Die `FraudDetection`-Klasse verfügt über folgende Attribute:

- `transaction`: Eine Referenz auf die Transaktion, die auf Betrug analysiert wird.

- `risk_factors`: Ein Array, das die mit der Transaktion verbundenen Risikofaktoren speichert. Jeder Risikofaktor wird als Hash dargestellt, wobei der Schlüssel die Beschreibung des Risikofaktors ist und der Wert die mit diesem Risikofaktor verbundene Betrugswahrscheinlichkeit.

Die Methode `add_risk_factor` ermöglicht das Hinzufügen eines Risikofaktors zum `risk_factors`-Array. Sie akzeptiert zwei Parameter: `description`, einen String, der den Risikofaktor beschreibt, und `probability`, einen Float-Wert, der die mit diesem Risikofaktor verbundene Betrugswahrscheinlichkeit darstellt. Wir verwenden eine `case..in`-Bedingung für eine einfache Typüberprüfung.

Die Methode `high_risk?`, die am Ende der Kette überprüft wird, ist eine Prädikatmethode, die die `fraud_probability` (berechnet durch die Summe aller Risikofaktorwahrscheinlichkeiten) mit dem `RISK_THRESHOLD` vergleicht.

Die `FraudDetection`-Klasse bietet eine saubere und gekapselte Möglichkeit zur Verwaltung der Betrugserkennung für eine Transaktion. Sie ermöglicht das Hinzufügen mehrerer Risikofaktoren, jeweils mit eigener Beschreibung und Wahrscheinlichkeit, und stellt eine Methode bereit, um festzustellen, ob die Transaktion basierend auf der berechneten Betrugswahrscheinlichkeit als hochriskant eingestuft wird. Die Klasse kann problemlos in ein größeres Betrugserkennungssystem integriert werden, bei dem verschiedene Komponenten zusammenarbeiten, um das Risiko betrügerischer Transaktionen zu bewerten und zu minimieren.

Da dies schließlich ein Buch über Programmierung mit KI ist, hier ein Beispiel für die Implementierung der `CheckCustomerHistory`-Klasse, die KI-Verarbeitung unter Verwendung des `ChatCompletion`-Moduls meiner [Raix](#)-Bibliothek nutzt:


```
1  class CheckCustomerHistory
2    include Raix::ChatCompletion
3
4    attr_accessor :fraud_detection
5
6    INSTRUCTION = <<~END
7      You are an AI assistant tasked with checking a customer's transaction
8      history for potential fraud indicators. Given the current transaction
9      and the customer's past transactions, analyze the data to identify any
10     suspicious patterns or anomalies.
11
12     Consider factors such as the frequency of transactions, transaction
13     amounts, geographical locations, and any deviations from the customer's
14     typical behavior to generate a probability score as a float in the range
15     of 0 to 1 (with 1 being absolute certainty of fraud).
16
17     Output the results of your analysis, highlighting any red flags or areas
18     of concern in the following JSON format:
19
20     { description: <Summary of your findings>, probability: <Float> }
21   END
22
23   def self.check(fraud_detection)
24     new(fraud_detection).call
25   end
26
27   def call
28     chat_completion(json: true).tap do |result|
29       fraud_detection.add_risk_factor(**result)
30     end
31     Result.ok(fraud_detection)
32   rescue StandardError => e
33     Result.err(FraudDetectionError.new(e))
34   end
35
36   private
37
38   def initialize(fraud_detection)
39     self.fraud_detection = fraud_detection
40   end
41
42   def transcript
```

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

In diesem Beispiel definiert `CheckCustomerHistory` eine `INSTRUCTION`-Konstante, die der KI spezifische Anweisungen zur Analyse des Transaktionsverlaufs des Kunden auf potenzielle Betrugshinweise mittels einer Systemanweisung bereitstellt.

Die `self.check`-Methode ist eine Klassenmethode, die eine neue Instanz von `CheckCustomerHistory` mit dem `fraud_detection`-Objekt initialisiert und die `call`-Methode aufruft, um die Kundenhistorieanalyse durchzuführen.

Innerhalb der `call`-Methode wird der Transaktionsverlauf des Kunden abgerufen und in ein Transkript formatiert, das an das KI-Modell übergeben wird. Das KI-Modell analysiert den Transaktionsverlauf basierend auf den bereitgestellten Anweisungen und gibt eine Zusammenfassung seiner Erkenntnisse zurück.

Die Erkenntnisse werden dem `fraud_detection`-Objekt hinzugefügt, und das aktualisierte `fraud_detection`-Objekt wird als erfolgreiches `Result` zurückgegeben.

Durch die Nutzung des `ChatCompletion`-Moduls kann die `CheckCustomerHistory`-Klasse die Leistungsfähigkeit der KI nutzen, um den Transaktionsverlauf des Kunden zu analysieren und potenzielle Betrugshinweise zu identifizieren. Dies ermöglicht sophisticatedere und adaptive Betrugserkennung, da das KI-Modell neue Muster und Anomalien im Laufe der Zeit lernen und sich anpassen kann.

Der aktualisierte `FraudDetectionWorker` und die `CheckCustomerHistory`-Klasse zeigen, wie KI-Worker nahtlos integriert werden können und den Betrugserkennung-Prozess durch intelligente Analyse- und Entscheidungsfähigkeiten verbessern.

Kundensentimentanalyse

Hier ist ein weiteres ähnliches Beispiel dafür, wie Sie einen Worker zur Kundensentimentanalyse implementieren können. Diesmal mit weniger Erklärungen, da Sie allmählich verstehen sollten, wie diese Art der Programmierung funktioniert:

```
1  class CustomerSentimentAnalysisWorker
2    include Wisper::Publisher
3
4    def call(feedback)
5      Result.ok(feedback)
6        .and_then(PreprocessFeedback.method(:preprocess))
7        .map(PerformSentimentAnalysis.method(:analyze))
8        .map(ExtractKeyPhrases.method(:extract))
9        .map(IdentifyTrends.method(:identify))
10       .map(GenerateInsights.method(:generate)).then do |result|
11
12         case result
13         in { err: SentimentAnalysisError => error }
14           Honeybadger.notify(error.message, context: {feedback:})
15         in { ok: SentimentAnalysisResult => result }
16           broadcast(:sentiment_analysis_completed, result)
17         end
18       end
19     end
20   end
```

In diesem Beispiel umfassen die Schritte des CustomerSentimentAnalysisWorker die Vorverarbeitung des Feedbacks (z.B. Entfernung von Störungen, Tokenisierung), die Durchführung einer Stimmungsanalyse zur Bestimmung der allgemeinen Stimmung (positiv, negativ oder neutral), die Extraktion von Schlüsselphrasen und Themen, die Identifizierung von Trends und Mustern sowie die Generierung umsetzbarer Erkenntnisse auf Basis der Analyse.

Anwendungen im Gesundheitswesen

Im Gesundheitsbereich können KI-Worker medizinisches Fachpersonal und Forscher bei verschiedenen Aufgaben unterstützen, was zu verbesserten Patientenergebnissen und beschleunigten medizinischen Entdeckungen führt. Einige Beispiele sind:

Patientenaufnahme

KI-Worker können den Patientenaufnahmeprozess durch Automatisierung verschiedener Aufgaben und intelligente Unterstützung optimieren.

Terminplanung: KI-Worker können die Terminplanung übernehmen, indem sie Patientenpräferenzen, Verfügbarkeit und die Dringlichkeit ihrer medizinischen Bedürfnisse berücksichtigen. Sie können über Konversationsschnittstellen mit Patienten interagieren, sie durch den Planungsprozess führen und die am besten geeigneten Terminslots basierend auf den Anforderungen des Patienten und der Verfügbarkeit des Gesundheitsdienstleisters finden.

Erfassung der Krankengeschichte: Während der Patientenaufnahme können KI-Worker bei der Erfassung und Dokumentation der Krankengeschichte des Patienten unterstützen. Sie können interaktive Dialoge mit Patienten führen und relevante Fragen zu früheren Erkrankungen, Medikamenten, Allergien und Familienvorgeschichte stellen. Die KI-Worker können Techniken zur Verarbeitung natürlicher Sprache nutzen, um die gesammelten Informationen zu interpretieren und zu strukturieren und sicherzustellen, dass sie korrekt in der elektronischen Patientenakte erfasst werden.

Symptombeurteilung und Stratifizierung: KI-Worker können erste Symptombeurteilungen durchführen, indem sie Patienten nach ihren aktuellen Symptomen, deren Dauer, Schweregrad und damit verbundenen Faktoren befragen. Durch die Nutzung medizinischer Wissensdatenbanken und maschineller Lernmodelle können diese Worker die bereitgestellten Informationen analysieren und vorläufige

Differentialdiagnosen erstellen oder geeignete nächste Schritte empfehlen, wie etwa die Vereinbarung einer Konsultation mit einem Gesundheitsdienstleister oder die Empfehlung von Selbstpflegemaßnahmen.

Versicherungsüberprüfung: KI-Worker können bei der Versicherungsüberprüfung während der Patientenaufnahme unterstützen. Sie können Versicherungsdaten der Patienten sammeln, über APIs oder Webservices mit Versicherungsanbietern kommunizieren und die Anspruchsberechtigung und Leistungen überprüfen. Diese Automatisierung hilft dabei, den Versicherungsüberprüfungsprozess zu optimieren, den Verwaltungsaufwand zu reduzieren und eine genaue Informationserfassung sicherzustellen.

Patientenaufklärung und Anweisungen: KI-Worker können Patienten relevantes Aufklärungsmaterial und Anweisungen basierend auf ihren spezifischen Erkrankungen oder bevorstehenden Eingriffen zur Verfügung stellen. Sie können personalisierte Inhalte bereitstellen, häufige Fragen beantworten und Anleitungen zu Vorbereitungen vor dem Termin, Medikamentenanweisungen oder Nachsorge geben. Dies hilft, Patienten während ihrer gesamten Gesundheitsreise informiert und engagiert zu halten.

Durch den Einsatz von KI-Workern bei der Patientenaufnahme können Gesundheitsorganisationen die Effizienz steigern, Wartezeiten reduzieren und das gesamte Patientenerlebnis verbessern. Diese Worker können Routineaufgaben übernehmen, genaue Informationen sammeln und personalisierte Unterstützung bieten, sodass sich das Gesundheitspersonal auf die Bereitstellung hochwertiger Patientenversorgung konzentrieren kann.

Patientenrisikobewertung

KI-Worker können eine entscheidende Rolle bei der Bewertung von Patientenrisiken spielen, indem sie verschiedene Datenquellen analysieren und fortgeschrittene Analysetechniken anwenden.

Datenintegration: KI-Worker können Patientendaten aus verschiedenen Quellen sammeln und auswerten, wie etwa elektronische Patientenakten, medizinische Bildgebung, Laborergebnisse, tragbare Geräte und soziale Gesundheitsdeterminanten. Durch die Zusammenführung dieser Informationen zu einem umfassenden Patientenprofil können KI-Worker einen ganzheitlichen Überblick über den Gesundheitszustand und die Risikofaktoren des Patienten bieten.

Risikostratifizierung: KI-Worker können prädiktive Modelle verwenden, um Patienten basierend auf ihren individuellen Merkmalen und Gesundheitsdaten in verschiedene Risikokategorien einzuteilen. Diese Risikostratifizierung ermöglicht es Gesundheitsdienstleistern, Patienten zu priorisieren, die eine unmittelbare Aufmerksamkeit oder Intervention benötigen. Beispielsweise können Patienten, die als Hochrisikopatienten für eine bestimmte Erkrankung identifiziert wurden, für eine engere Überwachung, präventive Maßnahmen oder frühzeitige Intervention gekennzeichnet werden.

Personalisierte Risikoprofile: KI-Worker können für jeden Patienten personalisierte Risikoprofile erstellen, die die spezifischen Faktoren hervorheben, die zu ihren Risikobewertungen beitragen. Diese Profile können Einblicke in den Lebensstil des Patienten, genetische Veranlagungen, Umweltfaktoren und soziale Gesundheitsdeterminanten enthalten. Durch die Bereitstellung einer detaillierten Aufschlüsselung der Risikofaktoren können KI-Worker Gesundheitsdienstleistern helfen, Präventionsstrategien und Behandlungspläne auf die individuellen Patientenbedürfnisse abzustimmen.

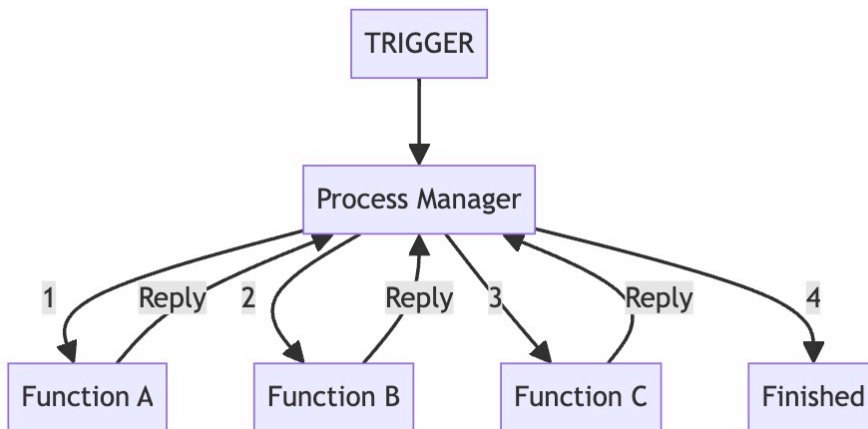
Kontinuierliche Risikoüberwachung: KI-Worker können Patientendaten kontinuierlich überwachen und Risikobewertungen in Echtzeit aktualisieren. Wenn neue Informationen verfügbar werden, wie etwa Änderungen der Vitalzeichen, Laborergebnisse oder Medikamentenadhärenz, können KI-Worker Risikobewertungen neu berechnen und Gesundheitsdienstleister über signifikante Änderungen informieren. Diese proaktive Überwachung ermöglicht zeitnahe Interventionen und Anpassungen

der Patientenversorgungspläne.

Klinische Entscheidungsunterstützung: KI-Worker können Risikobewertungsergebnisse in klinische Entscheidungsunterstützungssysteme integrieren und Gesundheitsdienstleistern evidenzbasierte Empfehlungen und Warnungen bereitstellen. Wenn beispielsweise der Risikowert eines Patienten für eine bestimmte Erkrankung einen bestimmten Schwellenwert überschreitet, kann der KI-Worker den Gesundheitsdienstleister auffordern, bestimmte diagnostische Tests, präventive Maßnahmen oder Behandlungsoptionen basierend auf klinischen Leitlinien und bewährten Praktiken in Betracht zu ziehen.

Diese Worker können riesige Mengen an Patientendaten verarbeiten, anspruchsvolle Analysen durchführen und verwertbare Erkenntnisse zur Unterstützung klinischer Entscheidungsfindung generieren. Dies führt letztendlich zu verbesserten Patientenergebnissen, reduzierten Gesundheitskosten und einem verbesserten Management der Bevölkerungsgesundheit.

KI-Worker als Prozess-Manager



Im Kontext KI-gesteuerter Anwendungen kann ein Worker als Process Manager konzipiert werden, wie im Buch “Enterprise Integration Patterns” von Gregor Hohpe beschrieben. Ein Process Manager ist eine zentrale Komponente, die den Status eines Prozesses verwaltet und basierend auf Zwischenergebnissen die nächsten Verarbeitungsschritte bestimmt.

Wenn ein KI-Worker als Process Manager agiert, empfängt er eine eingehende Nachricht, die den Prozess initialisiert, bekannt als *Triggernachricht*. Der KI-Worker verwaltet dann den Status der Prozessausführung (als Gesprächsprotokoll) und bearbeitet die Nachricht durch eine Reihe von Verarbeitungsschritten, die als Werkzeugfunktionen implementiert sind. Diese können sequentiell oder parallel ausgeführt werden und werden nach seinem Ermessen aufgerufen.



Wenn Sie eine KI-Modellklasse wie GPT-4 verwenden, die weiß, wie man Funktionen parallel ausführt, dann kann Ihr Worker mehrere Schritte gleichzeitig ausführen. Zugegeben, ich habe das selbst noch nicht versucht, und mein Bauchgefühl sagt mir, dass die Ergebnisse variieren können.

Nach jedem einzelnen Verarbeitungsschritt wird die Kontrolle zurück an den KI-Worker übergeben, wodurch er basierend auf dem aktuellen Status und den erhaltenen Ergebnissen die nächsten Verarbeitungsschritte bestimmen kann.

Speichern Sie Ihre Triggernachrichten

Meiner Erfahrung nach ist es klug, Ihre Triggernachricht als datenbankgestütztes Objekt zu implementieren. Auf diese Weise wird jede Prozessinstanz durch einen eindeutigen Primärschlüssel identifiziert und Sie haben einen Platz, um den mit der Ausführung verbundenen Status zu speichern, einschließlich des KI-Gesprächsprotokolls.

Hier ist beispielsweise eine vereinfachte Version von Olympias AccountChange-Modelklasse, die eine Anfrage zur Änderung eines Benutzerkontos darstellt.

```

1  # == Schema Information
2  #
3  # Table name: account_changes
4  #
5  #   id          :uuid          not null, primary key
6  #   description :string
7  #   state       :string        not null
8  #   transcript  :jsonb
9  #   created_at  :datetime      not null
10 #   updated_at  :datetime      not null
11 #   account_id  :uuid          not null
12 #
13 # Indexes
14 #
15 #   index_account_changes_on_account_id (account_id)
16 #
17 # Foreign Keys

```

```
18 #
19 # fk_rails_... (account_id => accounts.id)
20 #
21 class AccountChange < ApplicationRecord
22   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
```

Die AccountChange-Klasse dient als Trigger-Nachricht, die einen Prozess zur Behandlung der Kontoänderungsanfrage initiiert. Beachten Sie, wie sie an Olympias [Wisper](#)-basiertes Pub/Sub-Subsystem gesendet wird, nachdem die Erstellungstransaktion abgeschlossen ist.

Die Speicherung der Trigger-Nachricht in der Datenbank auf diese Weise bietet eine dauerhafte Aufzeichnung jeder Kontoänderungsanfrage. Jeder Instanz der AccountChange-Klasse wird ein eindeutiger Primärschlüssel zugewiesen, was eine einfache Identifizierung und Verfolgung einzelner Anfragen ermöglicht. Dies ist besonders nützlich für Audit-Logging-Zwecke, da das System dadurch einen historischen Überblick über alle Kontoänderungen behalten kann, einschließlich des Zeitpunkts der Anfrage, der gewünschten Änderungen und des aktuellen Status jeder Anfrage.

Im gegebenen Beispiel enthält die AccountChange-Klasse Felder wie `description` zur

Erfassung der Details der angeforderten Änderung, `state` zur Darstellung des aktuellen Status der Anfrage (z.B. angefordert, abgeschlossen, benötigt_manuelle_überprüfung) und `transcript` zur Speicherung des KI-Gesprächsprotokolls im Zusammenhang mit der Anfrage. Das `description`-Feld ist der eigentliche Prompt, der verwendet wird, um die erste Chat-Completion mit der KI zu initiieren. Die Speicherung dieser Daten bietet wertvollen Kontext und ermöglicht eine bessere Nachverfolgung und Analyse des Kontoänderungsprozesses.

Die Speicherung von Trigger-Nachrichten in der Datenbank ermöglicht eine robuste Fehlerbehandlung und Wiederherstellung. Wenn während der Verarbeitung einer Kontoänderungsanfrage ein Fehler auftritt, markiert das System die Anfrage als fehlgeschlagen und überführt sie in einen Status, der menschliches Eingreifen erfordert. Dies stellt sicher, dass keine Anfrage verloren geht oder vergessen wird und alle Probleme ordnungsgemäß behandelt und gelöst werden können.

Der KI-Worker als Process Manager bietet einen zentralen Kontrollpunkt und ermöglicht leistungsfähige Prozessberichts- und Debugging-Funktionen. Es ist jedoch wichtig zu beachten, dass die Verwendung eines KI-Workers als Process Manager für jedes Workflow-Szenario in Ihrer Anwendung möglicherweise übertrieben sein könnte.

Integration von KI-Workern in Ihre Anwendungsarchitektur

Bei der Integration von KI-Workern in Ihre Anwendungsarchitektur müssen verschiedene technische Aspekte berücksichtigt werden, um eine reibungslose Integration und effektive Kommunikation zwischen den KI-Workern und anderen Anwendungskomponenten zu gewährleisten. Dieser Abschnitt behandelt wichtige Aspekte der Gestaltung dieser Schnittstellen, der Handhabung des Datenflusses und der Verwaltung des Lebenszyklus von KI-Workern.

Gestaltung klarer Schnittstellen und Kommunikationsprotokolle

Um eine nahtlose Integration zwischen KI-Workern und anderen Anwendungskomponenten zu ermöglichen, ist es entscheidend, klare Schnittstellen und Kommunikationsprotokolle zu definieren. Berücksichtigen Sie die folgenden Ansätze:

API-basierte Integration: Stellen Sie die Funktionalität von KI-Workern über klar definierte APIs bereit, wie beispielsweise RESTful-Endpunkte oder GraphQL-Schemas. Dies ermöglicht anderen Komponenten die Interaktion mit den KI-Workern über Standard-HTTP-Anfragen und -Antworten. Die API-basierte Integration bietet einen klaren Vertrag zwischen den KI-Workern und den verbrauchenden Komponenten, was die Entwicklung, das Testen und die Wartung der Integrationspunkte erleichtert.

Nachrichtenbasierte Kommunikation: Implementieren Sie nachrichtenbasierte Kommunikationsmuster, wie Nachrichtenwarteschlangen oder Publish-Subscribe-Systeme, um eine asynchrone Interaktion zwischen KI-Workern und anderen Komponenten zu ermöglichen. Dieser Ansatz entkoppelt die KI-Worker vom Rest der Anwendung und ermöglicht eine bessere Skalierbarkeit, Fehlertoleranz und lose Kopplung. Nachrichtenbasierte Kommunikation ist besonders nützlich, wenn die Verarbeitung durch KI-Worker zeitaufwändig oder ressourcenintensiv ist, da sie anderen Teilen der Anwendung ermöglicht, ohne Wartezeit auf den Abschluss der KI-Worker-Aufgaben weiterzuarbeiten.

Ereignisgesteuerte Architektur: Gestalten Sie Ihr System um Ereignisse und Trigger herum, die KI-Worker aktivieren, wenn bestimmte Bedingungen erfüllt sind. KI-Worker können relevante Ereignisse abonnieren und entsprechend reagieren, indem sie ihre zugewiesenen Aufgaben ausführen, wenn die Ereignisse eintreten. Eine ereignisgesteuerte Architektur ermöglicht Echtzeit-Verarbeitung und erlaubt es, KI-Worker bei Bedarf aufzurufen, wodurch unnötiger Ressourcenverbrauch reduziert wird. Dieser Ansatz eignet sich gut für Szenarien, in denen KI-Worker auf bestimmte Aktionen oder Änderungen im Anwendungszustand reagieren müssen.

Umgang mit Datenfluss und Synchronisation

Bei der Integration von KI-Workern in Ihre Anwendung ist es entscheidend, einen reibungslosen Datenfluss sicherzustellen und die Datenkonsistenz zwischen den KI-Workern und anderen Komponenten aufrechtzuerhalten. Berücksichtigen Sie die folgenden Aspekte:

Datenvorbereitung: Bevor Daten in KI-Worker eingespeist werden, müssen möglicherweise verschiedene Datenverarbeitungsaufgaben durchgeführt werden, wie das Bereinigen, Formatieren und/oder Transformieren der Eingabedaten. Sie möchten nicht nur sicherstellen, dass die KI-Worker effektiv arbeiten können, sondern auch, dass Sie keine Token für Informationen verschwenden, die der Worker bestenfalls als nutzlos, schlimmstenfalls als störend empfinden könnte. Die Datenverarbeitung kann Aufgaben wie das Entfernen von Rauschen, den Umgang mit fehlenden Werten oder die Konvertierung von Datentypen umfassen.

Datenpersistenz: Wie werden Sie die Daten speichern und persistent halten, die in KI-Worker ein- und ausfließen? Berücksichtigen Sie Faktoren wie Datenvolumen, Abfragemuster und Skalierbarkeit. Müssen Sie das Transkript der KI als Reflexion ihres “Gedankenprozesses” für Audit- oder Debugging-Zwecke speichern, oder reicht es aus, nur eine Aufzeichnung der Ergebnisse zu haben?

Datenabruf: Das Abrufen der von Workern benötigten Daten kann Datenbankabfragen, Lesen aus Dateien oder Zugriff auf externe APIs umfassen. Berücksichtigen Sie die Latenzzeit und wie KI-Worker Zugriff auf die aktuellsten Daten erhalten. Benötigen sie vollen Zugriff auf Ihre Datenbank oder sollten Sie den Umfang ihres Zugriffs eng nach ihren Aufgaben definieren? Was ist mit der Skalierung? Erwägen Sie Caching-Mechanismen zur Verbesserung der Leistung und Reduzierung der Last auf die zugrunde liegenden Datenquellen.

Datensynchronisation: Wenn mehrere Komponenten, einschließlich KI-Worker, auf gemeinsame Daten zugreifen und diese ändern, ist es wichtig, geeignete

Synchronisationsmechanismen zu implementieren, um die Datenkonsistenz zu gewährleisten. Datenbank-Sperrmechanismen, wie optimistisches oder pessimistisches Sperren, können dabei helfen, Konflikte zu vermeiden und die Datenintegrität sicherzustellen. Implementieren Sie Transaktionsmanagement-Techniken, um zusammenhängende Datenoperationen zu gruppieren und die ACID-Eigenschaften (Atomarität, Konsistenz, Isolation und Dauerhaftigkeit) zu gewährleisten.

Fehlerbehandlung und Wiederherstellung: Implementieren Sie robuste Fehlerbehandlungs- und Wiederherstellungsmechanismen, um mit datenbezogenen Problemen umzugehen, die während des Datenflussprozesses auftreten können. Behandeln Sie Ausnahmen elegant und stellen Sie aussagekräftige Fehlermeldungen zur Unterstützung der Fehlerbehebung bereit. Implementieren Sie Wiederholungsmechanismen und Fallback-Strategien, um temporäre Ausfälle oder Netzwerkunterbrechungen zu behandeln. Definieren Sie klare Verfahren für die Datenwiederherstellung im Fall von Datenbeschädigung oder -verlust.

Durch sorgfältiges Design und Implementierung von Datenfluss- und Synchronisationsmechanismen können Sie sicherstellen, dass Ihre KI-Worker Zugriff auf genaue, konsistente und aktuelle Daten haben. Dies ermöglicht ihnen, ihre Aufgaben effektiv auszuführen und zuverlässige Ergebnisse zu liefern.

Verwaltung des Lebenszyklus von KI-Workern

Entwickeln Sie einen standardisierten Prozess für die Initialisierung und Konfiguration von KI-Workern. Ich bevorzuge Frameworks, die standardisieren, wie Sie Einstellungen wie Modellnamen, Systemanweisungen und Funktionsdefinitionen festlegen. Stellen Sie sicher, dass der Initialisierungsprozess automatisiert und reproduzierbar ist, um die Bereitstellung und Skalierung zu erleichtern.

Implementieren Sie umfassende Überwachungs- und Protokollierungsmechanismen, um den Zustand und die Leistung von KI-Workern zu verfolgen. Erfassen Sie Metriken wie Ressourcennutzung, Verarbeitungszeit, Fehlerraten und Durchsatz. Verwenden

Sie zentralisierte Protokollierungssysteme wie den ELK-Stack (Elasticsearch, Logstash, Kibana), um Protokolle von mehreren KI-Workern zu aggregieren und zu analysieren.

Bauen Sie Fehlertoleranz und Resilienz in die KI-Worker-Architektur ein. Implementieren Sie Fehlerbehandlungs- und Wiederherstellungsmechanismen, um Ausfälle oder Ausnahmen elegant zu behandeln. Large Language Models sind noch Bleeding-Edge-Technologie; Anbieter fallen oft zu unerwarteten Zeiten aus. Verwenden Sie Wiederholungsmechanismen und Schutzschalter, um Kaskadenausfälle zu verhindern.

Komponierbarkeit und Orchestrierung von KI-Workern

Einer der wichtigsten Vorteile der KI-Worker-Architektur ist ihre Komponierbarkeit, die es ermöglicht, mehrere KI-Worker zu kombinieren und zu orchestrieren, um komplexe Probleme zu lösen. Indem Sie eine größere Aufgabe in kleinere, besser handhabbare Teilaufgaben aufteilen, die jeweils von einem spezialisierten KI-Worker bearbeitet werden, können Sie leistungsfähige und flexible Systeme erstellen. In diesem Abschnitt werden wir verschiedene Ansätze zur Komposition und Orchestrierung “einer Vielzahl” von KI-Workern untersuchen.

Verkettung von KI-Workern für mehrstufige Workflows

In vielen Szenarien kann eine komplexe Aufgabe in eine Reihe sequentieller Schritte zerlegt werden, wobei die Ausgabe eines KI-Workers zur Eingabe für den nächsten wird. Diese Verkettung von KI-Workern erstellt einen mehrstufigen Workflow oder eine Pipeline. Jeder KI-Worker in der Kette konzentriert sich auf eine spezifische Teilaufgabe, und die endgültige Ausgabe ist das Ergebnis der kombinierten Bemühungen aller Worker.

Betrachten wir ein Beispiel im Kontext einer Ruby on Rails-Anwendung zur Verarbeitung von nutzergenerierten Inhalten. Der Workflow umfasst die folgenden Schritte, die zugegebenermaßen in realen Anwendungsfällen wahrscheinlich jeweils zu einfach sind, um eine solche Zerlegung zu rechtfertigen, aber sie machen das Beispiel leichter verständlich:

1. **Textbereinigung:** Ein KI-Worker, der für das Entfernen von HTML-Tags, die Konvertierung in Kleinbuchstaben und die Handhabung der Unicode-Normalisierung zuständig ist.
2. **Spracherkennung:** Ein KI-Worker, der die Sprache des bereinigten Textes identifiziert.
3. **Stimmungsanalyse:** Ein KI-Worker, der die Stimmung (positiv, negativ oder neutral) des Textes basierend auf der erkannten Sprache bestimmt.
4. **Inhaltskategorisierung:** Ein KI-Worker, der den Text mithilfe von Techniken der natürlichen Sprachverarbeitung in vordefinierte Kategorien einordnet.

Hier ist ein sehr vereinfachtes Beispiel, wie Sie diese KI-Worker mit Ruby verketteten können:

```
1 class ContentProcessor
2   def initialize(text)
3     @text = text
4   end
5
6   def process
7     cleaned_text = TextCleanupWorker.new(@text).call
8     language = LanguageDetectionWorker.new(cleaned_text).call
9     sentiment = SentimentAnalysisWorker.new(cleaned_text, language).call
10    category = CategorizationWorker.new(cleaned_text, language).call
11
12    { cleaned_text:, language:, sentiment:, category: }
13  end
14 end
```

In diesem Beispiel initialisiert die ContentProcessor-Klasse mit dem Rohtext und

verkettet die AI-Worker in der `process`-Methode miteinander. Jeder AI-Worker führt seine spezifische Aufgabe aus und übergibt das Ergebnis an den nächsten Worker in der Kette. Die endgültige Ausgabe ist ein Hash, der den bereinigten Text, die erkannte Sprache, die Stimmung und die Inhaltskategorie enthält.

Parallelverarbeitung für unabhängige AI-Worker

Im vorherigen Beispiel sind die AI-Worker sequentiell verkettet, wobei jeder Worker den Text verarbeitet und das Ergebnis an den nächsten Worker weitergibt. Wenn Sie jedoch mehrere AI-Worker haben, die unabhängig voneinander mit derselben Eingabe arbeiten können, können Sie den Arbeitsablauf optimieren, indem Sie sie parallel verarbeiten.

Im gegebenen Szenario können nach der Textbereinigung durch den `TextCleanupWorker` die `WorkerLanguageDetectionWorker`, `SentimentAnalysisWorker` und `CategorizationWorker` alle den bereinigten Text unabhängig voneinander verarbeiten. Durch die parallele Ausführung dieser Worker können Sie potenziell die gesamte Verarbeitungszeit reduzieren und die Effizienz Ihres Arbeitsablaufs verbessern.

Um Parallelverarbeitung in Ruby zu erreichen, können Sie Nebenläufigkeitstechniken wie Threads oder asynchrone Programmierung nutzen. Hier ist ein Beispiel, wie Sie die `ContentProcessor`-Klasse modifizieren können, um die letzten drei Worker parallel mithilfe von Threads zu verarbeiten:

```
1  require 'concurrent'
2
3  class ContentProcessor
4    def initialize(text)
5      @text = text
6    end
7
8    def process
9      cleaned_text = TextCleanupWorker.new(@text).call
10
11      language_future = Concurrent::Future.execute do
12        LanguageDetectionWorker.new(cleaned_text).call
13      end
14
15      sentiment_future = Concurrent::Future.execute do
16        SentimentAnalysisWorker.new(cleaned_text).call
17      end
18
19      category_future = Concurrent::Future.execute do
20        CategorizationWorker.new(cleaned_text).call
21      end
22
23      language = language_future.value
24      sentiment = sentiment_future.value
25      category = category_future.value
26
27      { cleaned_text:, language:, sentiment:, category: }
28    end
29  end
```

In dieser optimierten Version verwenden wir die `concurrent-ruby`-Bibliothek, um `Concurrent::Future`-Objekte für jeden der unabhängigen KI-Worker zu erstellen. Ein `Future` repräsentiert eine Berechnung, die asynchron in einem separaten Thread ausgeführt wird.

Nach dem Text-Bereinigungsschritt erstellen wir drei `Future`-Objekte: `language_future`, `sentiment_future` und `category_future`. Jedes `Future` führt seinen entsprechenden KI-Worker (`LanguageDetectionWorker`, `SentimentAnalysisWorker` und `CategorizationWorker`) in einem separaten

Thread aus und übergibt dabei den `cleaned_text` als Eingabe.

Durch den Aufruf der `value`-Methode auf jedem `Future` warten wir auf den Abschluss der Berechnung und erhalten das Ergebnis. Die `value`-Methode blockiert, bis das Ergebnis verfügbar ist, und stellt damit sicher, dass alle parallelen Worker ihre Verarbeitung abgeschlossen haben, bevor fortgefahren wird.

Schließlich erstellen wir den Ausgabe-Hash mit dem bereinigten Text und den Ergebnissen der parallelen Worker, genau wie im ursprünglichen Beispiel.

Durch die parallele Verarbeitung der unabhängigen KI-Worker können Sie die gesamte Verarbeitungszeit im Vergleich zur sequenziellen Ausführung möglicherweise reduzieren. Diese Optimierung ist besonders vorteilhaft bei zeitaufwändigen Aufgaben oder bei der Verarbeitung großer Datenmengen.

Es ist jedoch wichtig zu beachten, dass die tatsächlichen Leistungsgewinne von verschiedenen Faktoren abhängen, wie der Komplexität jedes Workers, den verfügbaren Systemressourcen und dem Overhead des Thread-Managements. Es ist immer eine gute Praxis, Ihr Code zu benchmarken und zu profilieren, um den optimalen Grad der Parallelisierung für Ihren spezifischen Anwendungsfall zu ermitteln.

Achten Sie außerdem bei der Implementierung der parallelen Verarbeitung auf gemeinsam genutzte Ressourcen oder Abhängigkeiten zwischen den Workern. Stellen Sie sicher, dass die Worker unabhängig voneinander arbeiten können, ohne Konflikte oder Race Conditions zu verursachen. Bei Abhängigkeiten oder gemeinsam genutzten Ressourcen müssen Sie möglicherweise geeignete Synchronisationsmechanismen implementieren, um die Datenintegrität zu gewährleisten und Probleme wie Deadlocks oder inkonsistente Ergebnisse zu vermeiden.

Rubys Global Interpreter Lock und asynchrone Verarbeitung

Es ist wichtig, die Auswirkungen von Rubys Global Interpreter Lock (GIL) zu verstehen, wenn man asynchrone Thread-basierte Verarbeitung in Ruby in Betracht zieht.

Der GIL ist ein Mechanismus im Ruby-Interpreter, der sicherstellt, dass nur ein Thread gleichzeitig Ruby-Code ausführen kann, selbst auf Mehrkern-Prozessoren. Das bedeutet, dass zwar mehrere Threads innerhalb eines Ruby-Prozesses erstellt und verwaltet werden können, aber nur ein Thread zu einem bestimmten Zeitpunkt aktiv Ruby-Code ausführen kann.

Der GIL wurde entwickelt, um die Implementierung des Ruby-Interpreters zu vereinfachen und Threadsicherheit für Rubys interne Datenstrukturen zu gewährleisten. Allerdings begrenzt er auch das Potenzial für echte parallele Ausführung von Ruby-Code.

Wenn Sie Threads in Ruby verwenden, wie zum Beispiel mit der `concurrent-ruby`-Bibliothek oder der eingebauten `Thread`-Klasse, unterliegen die Threads den Einschränkungen des GIL. Der GIL erlaubt jedem Thread, Ruby-Code für eine kurze Zeitscheibe auszuführen, bevor er zu einem anderen Thread wechselt, was die Illusion einer gleichzeitigen Ausführung erzeugt.

Aufgrund des GIL bleibt die tatsächliche Ausführung von Ruby-Code jedoch sequenziell. Während ein Thread Ruby-Code ausführt, sind andere Threads im Wesentlichen pausiert und warten darauf, dass sie an der Reihe sind, den GIL zu erhalten und ausgeführt zu werden.

Dies bedeutet, dass Thread-basierte asynchrone Verarbeitung in Ruby am effektivsten für I/O-gebundene Aufgaben ist, wie das Warten auf externe API-Antworten (wie von extern gehosteten Large Language Models) oder das Ausführen von Datei-I/O-Operationen. Wenn ein Thread auf eine I/O-Operation trifft, kann er den GIL freigeben und anderen Threads die Ausführung ermöglichen, während er auf den Abschluss der I/O wartet.

Andererseits kann der GIL bei CPU-gebundenen Aufgaben, wie intensiven Berechnungen oder lang laufender KI-Worker-Verarbeitung, die potenziellen Leistungsgewinne der Thread-basierten Parallelisierung einschränken. Da nur ein Thread gleichzeitig Ruby-Code ausführen kann, wird die Gesamtausführungszeit möglicherweise nicht signifikant im Vergleich zur sequenziellen Verarbeitung reduziert.

Um eine echte parallele Ausführung für CPU-gebundene Aufgaben in Ruby zu erreichen, müssen Sie möglicherweise alternative Ansätze in Betracht ziehen, wie zum Beispiel:

- Verwendung von prozessbasierter Parallelität mit mehreren Ruby-Prozessen, die jeweils auf einem separaten CPU-Kern laufen.
- Nutzung externer Bibliotheken oder Frameworks, die native Erweiterungen oder Schnittstellen zu Sprachen ohne GIL bereitstellen, wie C oder Rust.,
- Einsatz von verteilten Computing-Frameworks oder Message Queues, um Aufgaben über mehrere Maschinen oder Prozesse zu verteilen.

Es ist entscheidend, die Art Ihrer Aufgaben und die durch den GIL auferlegten Einschränkungen zu berücksichtigen, wenn Sie asynchrone Verarbeitung in Ruby entwerfen und implementieren. Während Thread-basierte asynchrone Verarbeitung Vorteile für I/O-gebundene Aufgaben bieten kann, bietet sie aufgrund der Einschränkungen des GIL möglicherweise keine signifikanten Leistungsverbesserungen für CPU-gebundene Aufgaben.

Ensemble-Techniken für verbesserte Genauigkeit

Ensemble-Techniken beinhalten die Kombination der Ausgaben mehrerer KI-Worker, um die Gesamtgenauigkeit oder Robustheit des Systems zu verbessern. Anstatt sich auf einen einzelnen KI-Worker zu verlassen, nutzen Ensemble-Techniken die kollektive

Intelligenz mehrerer Worker, um fundiertere Entscheidungen zu treffen.



Ensembles sind besonders wichtig, wenn verschiedene Teile Ihres Workflows am besten mit unterschiedlichen KI-Modellen funktionieren, was häufiger vorkommt, als Sie vielleicht denken. Leistungsstarke Modelle wie GPT-4 sind im Vergleich zu weniger leistungsfähigen Open-Source-Optionen extrem teuer und werden wahrscheinlich nicht für jeden einzelnen Workflow-Schritt Ihrer Anwendung benötigt.

Eine häufig verwendete Ensemble-Technik ist die Mehrheitsentscheidung, bei der mehrere KI-Worker unabhängig voneinander dieselbe Eingabe verarbeiten und die endgültige Ausgabe durch den Mehrheitskonsens bestimmt wird. Dieser Ansatz kann dazu beitragen, die Auswirkungen von Fehlern einzelner Worker zu minimieren und die Gesamtzuverlässigkeit des Systems zu verbessern.

Betrachten wir ein Beispiel, bei dem wir drei KI-Worker für die Stimmungsanalyse haben, die jeweils ein unterschiedliches Modell verwenden oder mit verschiedenen Kontexten ausgestattet sind. Wir können ihre Ausgaben mittels Mehrheitsentscheidung kombinieren, um die endgültige Stimmungsvorhersage zu bestimmen.

```
1 class SentimentAnalysisEnsemble
2   def initialize(text)
3     @text = text
4   end
5
6   def analyze
7     predictions = [
8       SentimentAnalysisWorker1.new(@text).analyze,
9       SentimentAnalysisWorker2.new(@text).analyze,
10      SentimentAnalysisWorker3.new(@text).analyze
11    ]
12
13    predictions
14      .group_by { |sentiment| sentiment }
15      .max_by { |_, votes| votes.size }
16      .first
```

```
17  
18     end  
19 end
```

In diesem Beispiel initialisiert die Klasse `SentimentAnalysisEnsemble` mit dem Text und ruft drei verschiedene KI-Worker für die Stimmungsanalyse auf. Die Methode `analyze` sammelt die Vorhersagen von jedem Worker und bestimmt die mehrheitliche Stimmung mithilfe der Methoden `group_by` und `max_by`. Das endgültige Ergebnis ist die Stimmung, die die meisten Stimmen vom Ensemble der Worker erhält.



Ensembles sind eindeutig ein Fall, bei dem sich Experimente mit Parallelisierung lohnen können.

Dynamische Auswahl und Aufruf von KI-Workern

In einigen, wenn nicht sogar den meisten Fällen, kann die Auswahl des spezifischen KI-Workers von Laufzeitbedingungen oder Benutzereingaben abhängen. Die dynamische Auswahl und Aufruf von KI-Workern ermöglicht Flexibilität und Anpassungsfähigkeit im System.



Möglicherweise werden Sie versucht sein, viel Funktionalität in einen einzigen KI-Worker zu packen, ihm viele Funktionen und einen großen, komplizierten Prompt zu geben, der erklärt, wie man sie aufruft. Widerstehen Sie dieser Versuchung, vertrauen Sie mir. Einer der Gründe, warum der Ansatz, den wir in diesem Kapitel besprechen, “Vielzahl von Workern” genannt wird, ist, uns daran zu erinnern, dass es wünschenswert ist, viele spezialisierte Worker zu haben, die jeweils ihre kleine Aufgabe im Dienste des größeren Zwecks erfüllen.

Betrachten Sie zum Beispiel eine Chatbot-Anwendung, bei der verschiedene KI-Worker für die Bearbeitung unterschiedlicher Arten von Benutzeranfragen zuständig sind.

Basierend auf der Benutzereingabe wählt die Anwendung dynamisch den geeigneten KI-Worker aus, um die Anfrage zu verarbeiten.

```
1 class ChatbotController < ApplicationController
2   def process_query
3     query = params[:query]
4     query_type = QueryClassifierWorker.new(query).classify
5
6     case query_type
7     when 'greeting'
8       response = GreetingWorker.new(query).generate_response
9     when 'product_inquiry'
10      response = ProductInquiryWorker.new(query).generate_response
11    when 'order_status'
12      response = OrderStatusWorker.new(query).generate_response
13    else
14      response = DefaultResponseWorker.new(query).generate_response
15    end
16
17    render json: { response: response }
18  end
19 end
```

In diesem Beispiel empfängt der ChatbotController eine Benutzeranfrage durch die process_query-Aktion. Zunächst verwendet er einen QueryClassifierWorker, um den Typ der Anfrage zu bestimmen. Basierend auf dem klassifizierten Anfragetyp wählt der Controller dynamisch den passenden KI-Worker aus, um die Antwort zu generieren. Diese dynamische Auswahl ermöglicht es dem Chatbot, verschiedene Arten von Anfragen zu verarbeiten und sie an die relevanten KI-Worker weiterzuleiten.



Da die Arbeit des `QueryClassifierWorker` relativ einfach ist und nicht viel Kontext oder Funktionsdefinitionen erfordert, können Sie ihn wahrscheinlich mit einem ultraschnellen kleinen LLM wie `mistralai/mixtral-8x7b-instruct:nitro` implementieren. Es verfügt über Fähigkeiten, die bei vielen Aufgaben nahe an das GPT-4-Niveau heranreichen, und zum Zeitpunkt, als ich dies schreibe, kann Groq es mit einem beeindruckenden Durchsatz von 444 Token pro Sekunde bereitstellen.

Kombination von traditionellem NLP mit LLMs

Während Large Language Models (LLMs) das Gebiet der Verarbeitung natürlicher Sprache (NLP) revolutioniert haben und eine unvergleichliche Vielseitigkeit und Leistung bei einer Vielzahl von Aufgaben bieten, sind sie nicht immer die effizienteste oder kostengünstigste Lösung für jedes Problem. In vielen Fällen kann die Kombination traditioneller NLP-Techniken mit LLMs zu optimierteren, gezielteren und wirtschaftlicheren Ansätzen zur Lösung spezifischer NLP-Herausforderungen führen.

Stellen Sie sich LLMs als Schweizer Taschenmesser des NLP vor - unglaublich vielseitig und leistungsstark, aber nicht unbedingt das beste Werkzeug für jede Aufgabe. Manchmal kann ein spezielles Werkzeug wie ein Korkenzieher oder ein Dosenöffner für eine bestimmte Aufgabe effektiver und effizienter sein. Ähnlich können traditionelle NLP-Techniken wie Dokumenten-Clustering, Themenidentifizierung und Klassifizierung oft gezieltere und kostengünstigere Lösungen für bestimmte Aspekte Ihrer NLP-Pipeline bieten.

Einer der wichtigsten Vorteile traditioneller NLP-Techniken ist ihre rechnerische Effizienz. Diese Methoden, die oft auf einfacheren statistischen Modellen oder regelbasierten Ansätzen basieren, können große Mengen an Textdaten viel schneller und mit geringerem Rechenaufwand verarbeiten als LLMs. Dies macht sie besonders gut

geeignet für Aufgaben, die die Analyse und Organisation großer Dokumentenkorpora umfassen, wie zum Beispiel das Clustering ähnlicher Artikel oder die Identifizierung wichtiger Themen innerhalb einer Textsammlung.

Darüber hinaus können traditionelle NLP-Techniken oft eine hohe Genauigkeit und Präzision für spezifische Aufgaben erreichen, besonders wenn sie mit domänenspezifischen Datensätzen trainiert wurden. Zum Beispiel kann ein gut abgestimmter Dokumentenklassifikator, der traditionelle maschinelle Lernalgorithmen wie Support-Vector-Maschinen (SVM) oder Naive-Bayes verwendet, Dokumente mit minimalem Rechenaufwand präzise in vordefinierte Kategorien einordnen.

LLMs glänzen jedoch besonders bei Aufgaben, die ein tieferes Verständnis von Sprache, Kontext und Argumentation erfordern. Ihre Fähigkeit, kohärenten und kontextuell relevanten Text zu generieren, Fragen zu beantworten und lange Passagen zusammenzufassen, ist mit traditionellen NLP-Methoden unerreichbar. LLMs können komplexe sprachliche Phänomene wie Mehrdeutigkeit, Koreferenz und idiomatische Ausdrücke effektiv handhaben, was sie für Aufgaben, die natürliche Sprachgenerierung oder -verständnis erfordern, unschätzbar macht.

Die wahre Stärke liegt in der Kombination traditioneller NLP-Techniken mit LLMs, um hybride Ansätze zu schaffen, die die Stärken beider nutzen. Durch die Verwendung traditioneller NLP-Methoden für Aufgaben wie Dokumentenvorverarbeitung, Clustering und Themenextraktion können Sie Ihre Textdaten effizient organisieren und strukturieren. Diese strukturierten Informationen können dann in LLMs für fortgeschrittenere Aufgaben eingespeist werden, wie das Generieren von Zusammenfassungen, das Beantworten von Fragen oder das Erstellen umfassender Berichte.

Betrachten wir zum Beispiel einen Anwendungsfall, bei dem Sie einen Trendbericht für einen bestimmten Bereich basierend auf einem großen Korpus einzelner Trenddokumente erstellen möchten. Anstatt sich ausschließlich auf LLMs zu verlassen, was für die Verarbeitung großer Textmengen rechenintensiv und zeitaufwändig sein

kann, können Sie einen hybriden Ansatz verwenden:

1. Verwenden Sie traditionelle NLP-Techniken wie Topic Modeling (z.B. Latente Dirichlet-Allokation) oder Clustering-Algorithmen (z.B. K-Means), um ähnliche Trenddokumente zu gruppieren und Schlüsselthemen innerhalb des Korpus zu identifizieren.
2. Speisen Sie die geclusterten Dokumente und identifizierten Themen in ein LLM ein und nutzen Sie dessen überlegene Sprachverständnis- und Generierungsfähigkeiten, um kohärente und informative Zusammenfassungen für jeden Cluster oder jedes Thema zu erstellen.
3. Verwenden Sie schließlich das LLM, um einen umfassenden Trendbericht zu generieren, indem Sie die einzelnen Zusammenfassungen kombinieren, die wichtigsten Trends hervorheben und Einblicke sowie Empfehlungen basierend auf den aggregierten Informationen bereitstellen.

Durch die Kombination traditioneller NLP-Techniken mit LLMs auf diese Weise können Sie große Mengen an Textdaten effizient verarbeiten, aussagekräftige Erkenntnisse gewinnen und qualitativ hochwertige Berichte generieren, während Sie gleichzeitig Rechenressourcen und Kosten optimieren.

Wenn Sie sich auf Ihre NLP-Projekte einlassen, ist es wichtig, die spezifischen Anforderungen und Einschränkungen jeder Aufgabe sorgfältig zu evaluieren und zu überlegen, wie traditionelle NLP-Methoden und LLMs gemeinsam genutzt werden können, um die besten Ergebnisse zu erzielen. Durch die Kombination der Effizienz und Präzision traditioneller Techniken mit der Vielseitigkeit und Leistungsfähigkeit von LLMs können Sie hocheffektive und wirtschaftliche NLP-Lösungen entwickeln, die Ihren Nutzern und Stakeholdern einen echten Mehrwert bieten.

Werkzeugnutzung



Im Bereich der KI-gesteuerten Anwendungsentwicklung hat sich das Konzept der “Werkzeugnutzung” oder des “Funktionsaufrufs” als leistungsstarke Technik etabliert, die es Ihrem LLM ermöglicht, sich mit externen Werkzeugen, APIs, Funktionen, Datenbanken und anderen Ressourcen zu verbinden. Dieser Ansatz ermöglicht ein reichhaltigeres Spektrum an Verhaltensweisen als die bloße Textausgabe und dynamischere Interaktionen zwischen Ihren KI-Komponenten und dem restlichen Ökosystem Ihrer Anwendung. Wie wir in diesem Kapitel untersuchen werden, bietet die Werkzeugnutzung auch die Möglichkeit, Ihr KI-Modell Daten auf strukturierte Weise generieren zu lassen.

Was ist Werkzeugnutzung?

Werkzeugnutzung, auch bekannt als Funktionsaufruf, ist eine Technik, die es Entwicklern ermöglicht, eine Liste von Funktionen zu definieren, mit denen ein LLM während des Generierungsprozesses interagieren kann. Diese Werkzeuge können von einfachen Hilfsfunktionen bis hin zu komplexen APIs oder Datenbankabfragen reichen. Indem sie dem LLM Zugriff auf diese Werkzeuge gewähren, können Entwickler die Fähigkeiten des Modells erweitern und es in die Lage versetzen, Aufgaben auszuführen, die externes Wissen oder Aktionen erfordern.

Abbildung 8. Beispiel einer Funktionsdefinition für einen KI-Arbeiter, der Dokumente analysiert

```
1  FUNCTION = {
2      name: "save_analysis",
3      description: "Save analysis data for document",
4      parameters: {
5          type: "object",
6          properties: {
7              title: {
8                  type: "string",
9                  maxLength: 140
10             },
11             summary: {
12                 type: "string",
13                 description: "comprehensive multi-paragraph summary with
14                             overview and list of sections (if applicable)"
15             },
16             tags: {
17                 type: "array",
18                 items: {
19                     type: "string",
20                     description: "lowercase tags representing main themes
21                                 of the document"
22                 }
23             }
24         },
25         "required": %w[title summary tags]
26     }
```

27 } .freeze

Der Kerngedanke hinter dem Werkzeugeinsatz besteht darin, dem LLM die Fähigkeit zu geben, basierend auf der Benutzereingabe oder der jeweiligen Aufgabe dynamisch die geeigneten Werkzeuge auszuwählen und auszuführen. Anstatt sich ausschließlich auf das vortrainierte Wissen des Modells zu verlassen, ermöglicht der Werkzeugeinsatz dem LLM, externe Ressourcen zu nutzen, um genauere, relevantere und handlungsorientierte Antworten zu generieren. Der Werkzeugeinsatz macht Techniken wie RAG (Retrieval Augmented Generation) deutlich einfacher zu implementieren, als sie es sonst wären.

Beachten Sie, dass dieses Buch, sofern nicht anders angegeben, davon ausgeht, dass Ihr KI-Modell keinen Zugriff auf integrierte serverseitige Werkzeuge hat. Alle Werkzeuge, die Sie Ihrer KI zur Verfügung stellen möchten, müssen von Ihnen explizit in jeder API-Anfrage deklariert werden, einschließlich Vorkehrungen für deren Ausführung, falls und wenn Ihre KI Ihnen mitteilt, dass sie dieses Werkzeug in ihrer Antwort verwenden möchte.

Das Potenzial des Werkzeugeinsatzes

Der Werkzeugeinsatz eröffnet ein breites Spektrum an Möglichkeiten für KI-gesteuerte Anwendungen. Hier sind einige Beispiele für das, was mit Werkzeugeinsatz erreicht werden kann:

1. **Chatbots und Virtuelle Assistenten:** Durch die Verbindung eines LLM mit externen Werkzeugen können Chatbots und virtuelle Assistenten komplexere Aufgaben ausführen, wie zum Beispiel Informationen aus Datenbanken abrufen, API-Aufrufe ausführen oder mit anderen Systemen interagieren. Ein Chatbot

könnte beispielsweise ein CRM-Werkzeug verwenden, um den Status eines Geschäfts basierend auf der Benutzeranfrage zu ändern.

2. **Datenanalyse und Erkenntnisse:** LLMs können mit Datenanalyse-Werkzeugen oder -Bibliotheken verbunden werden, um fortgeschrittene Datenverarbeitungsaufgaben durchzuführen. Dies ermöglicht Anwendungen, Erkenntnisse zu generieren, vergleichende Analysen durchzuführen oder datengesteuerte Empfehlungen basierend auf Benutzeranfragen bereitzustellen.
3. **Suche und Informationsabruf:** Der Werkzeugeinsatz ermöglicht LLMs die Interaktion mit Suchmaschinen, Vektordatenbanken oder anderen Informationsabrufsystemen. Durch die Umwandlung von Benutzeranfragen in Suchanfragen kann das LLM relevante Informationen aus mehreren Quellen abrufen und umfassende Antworten auf Benutzerfragen liefern.
4. **Integration mit externen Diensten:** Der Werkzeugeinsatz ermöglicht eine nahtlose Integration zwischen KI-gesteuerten Anwendungen und externen Diensten oder APIs. Ein LLM könnte beispielsweise mit einer Wetter-API interagieren, um Echtzeit-Wetterupdates bereitzustellen, oder mit einer Übersetzungs-API, um mehrsprachige Antworten zu generieren.

Der Werkzeugeinsatz-Workflow

Der Werkzeugeinsatz-Workflow umfasst typischerweise vier Hauptschritte:

1. Einbindung von Funktionsdefinitionen in Ihren Anfrage-Kontext
2. Dynamische (oder explizite) Werkzeugauswahl
3. Ausführung der Funktion(en)
4. Optionale Fortsetzung des ursprünglichen Prompts

Lassen Sie uns jeden dieser Schritte im Detail betrachten.

Einbindung von Funktionsdefinitionen in Ihren Anfrage-Kontext

Die KI weiß, welche Werkzeuge ihr zur Verfügung stehen, weil Sie ihr eine Liste als Teil Ihrer Completion-Anfrage übergeben (typischerweise definiert als Funktionen unter Verwendung einer Variante des JSON-Schemas).

Die genaue Syntax der Werkzeugdefinition ist modellspezifisch.

So definieren Sie eine `get_weather`-Funktion in Claude 3:

```
1  {
2      "name": "get_weather",
3      "description": "Get the current weather in a given location",
4      "input_schema": {
5          "type": "object",
6          "properties": {
7              "location": {
8                  "type": "string",
9                  "description": "The city and state, e.g. San Francisco, CA"
10             },
11             "unit": {
12                 "type": "string",
13                 "enum": ["celsius", "fahrenheit"],
14                 "description": "The unit of temperature"
15             }
16         },
17         "required": ["location"]
18     }
19 }
```

Und so würdest du dieselbe Funktion für GPT-4 definieren, indem du sie als Wert des `tools`-Parameters übergibst:


```
1  {
2    "name": "get_current_weather",
3    "description": "Get the current weather in a given location",
4    "parameters": {
5      "type": "object",
6      "properties": {
7        "location": {
8          "type": "string",
9          "description": "The city and state, e.g. San Francisco, CA",
10        },
11        "unit": {
12          "type": "string",
13          "enum": ["celsius", "fahrenheit"],
14          "description": "The unit of temperature"
15        },
16      },
17      "required": ["location"],
18    },
19  }
```

Fast gleich, aber ohne ersichtlichen Grund trotzdem anders! Wie ärgerlich.

Funktionsdefinitionen legen Name, Beschreibung und Eingabeparameter fest. Eingabeparameter können durch Attribute weiter definiert werden, zum Beispiel durch Enums zur Einschränkung der zulässigen Werte, sowie durch die Angabe, ob ein Parameter erforderlich ist oder nicht.

Zusätzlich zu den eigentlichen Funktionsdefinitionen können Sie auch Anweisungen oder Kontext dafür einbinden, warum und wie die Funktion in der Systemdirektive verwendet werden soll.

Zum Beispiel enthält mein Web-Suchwerkzeug in Olympia diese Systemdirektive, die die KI daran erinnert, dass sie über die erwähnten Werkzeuge verfügt:

```
1 The `google_search` and `realtime_search` functions let you do research
2 on behalf of the user. In contrast to Google, realtime search is powered
3 by Perplexity and provides real-time information to curated current events
4 databases and news sources. Make sure to include URLs in your response so
5 user can do followup research.
```

Die Bereitstellung detaillierter Beschreibungen gilt als wichtigster Faktor für die Leistung eines Tools. Ihre Beschreibungen sollten jedes Detail über das Tool erläutern, einschließlich:

- Was das Tool leistet
- Wann es eingesetzt werden sollte (und wann nicht)
- Was jeder Parameter bedeutet und wie er das Verhalten des Tools beeinflusst
- Alle wichtigen Vorbehalte oder Einschränkungen, die für die Implementierung des Tools gelten

Je mehr Kontext Sie der KI über Ihre Tools geben können, desto besser wird sie bei der Entscheidung sein, wann und wie sie diese einsetzen soll. Beispielsweise empfiehlt Anthropic für seine Claude 3-Serie mindestens 3-4 Sätze pro Tool-Beschreibung, bei komplexeren Tools auch mehr.

Es mag nicht unbedingt intuitiv sein, aber Beschreibungen werden auch als wichtiger erachtet als Beispiele. Während Sie in der Beschreibung eines Tools oder im begleitenden Prompt Beispiele für dessen Verwendung aufnehmen können, ist dies weniger wichtig als eine klare und umfassende Erklärung des Zwecks und der Parameter des Tools. Fügen Sie Beispiele erst hinzu, nachdem Sie die Beschreibung vollständig ausgearbeitet haben.

Hier ist ein Beispiel für eine Stripe-ähnliche API-Funktionsspezifikation:

```
1  {
2    "name": "createPayment",
3    "description": "Create a new payment request",
4    "parameters": {
5      "type": "object",
6      "properties": {
7        "transaction_amount": {
8          "type": "number",
9          "description": "The amount to be paid"
10       },
11       "description": {
12         "type": "string",
13         "description": "A brief description of the payment"
14       },
15       "payment_method_id": {
16         "type": "string",
17         "description": "The payment method to be used"
18       },
19       "payer": {
20         "type": "object",
21         "description": "Information about the payer, including their name,
22                        email, and identification number",
23         "properties": {
24           "name": {
25             "type": "string",
26             "description": "The payer's name"
27           },
28           "email": {
29             "type": "string",
30             "description": "The payer's email address"
31           },
32           "identification": {
33             "type": "object",
34             "description": "The payer's identification number",
35             "properties": {
36               "type": {
37                 "type": "string",
38                 "description": "Identification document (e.g. CPF, CNPJ)"
39               },
40               "number": {
41                 "type": "string",
42                 "description": "The identification number"
```

```
43         }
44     },
45     "required": [ "type", "number" ]
46 }
47 },
48 "required": [ "name", "email", "identification" ]
49 }
50 }
51 }
```



In der Praxis haben einige Modelle Schwierigkeiten im Umgang mit verschachtelten Funktionsspezifikationen und komplexen Ausgabedatentypen wie Arrays, Dictionaries usw. Theoretisch sollten Sie jedoch JSON-Schema-Spezifikationen beliebiger Tiefe bereitstellen können!

Dynamische Werkzeugauswahl

Wenn Sie eine Chat-Completion mit Werkzeugdefinitionen ausführen, wählt das LLM dynamisch die am besten geeigneten Werkzeuge aus und generiert die erforderlichen Eingangsparameter für jedes Werkzeug.

In der Praxis ist die Fähigkeit der KI, *genau* die richtige Funktion aufzurufen und *genau* Ihrer Spezifikation für die Eingaben zu folgen, nicht immer zuverlässig. Das Heruntersetzen des Temperatur-Hyperparameters auf 0,0 hilft erheblich, aber nach meiner Erfahrung werden Sie trotzdem gelegentlich Fehler sehen. Zu diesen Fehlern gehören halluzinierte Funktionsnamen, falsch benannte oder schlicht fehlende Eingangsparameter. Parameter werden als JSON übergeben, was bedeutet, dass Sie manchmal Fehler sehen werden, die durch abgeschnittenes, falsch zitiertes oder anderweitig fehlerhaftes JSON verursacht werden.



[Selbstheilende Daten](#)-Muster können helfen, [fehlerhafte JSON automatisch zu korrigieren](#) bei Funktionsaufrufen, die aufgrund von Syntaxfehlern fehlschlagen.

Erzwungene (bzw. Explizite) Werkzeugauswahl

Einige Modelle bieten die Möglichkeit, den Aufruf einer bestimmten Funktion als Parameter in der Anfrage zu erzwingen. Andernfalls liegt es vollständig im Ermessen der KI, ob die Funktion aufgerufen wird oder nicht.

Die Fähigkeit, einen Funktionsaufruf zu erzwingen, ist in bestimmten Szenarien entscheidend, in denen Sie sicherstellen möchten, dass ein spezifisches Werkzeug oder eine spezifische Funktion ausgeführt wird, unabhängig vom dynamischen Auswahlprozess der KI. Dafür gibt es mehrere wichtige Gründe:

1. **Explizite Kontrolle:** Möglicherweise verwenden Sie die KI als *Diskrete Komponente* oder in einem vordefinierten Workflow, der die Ausführung einer bestimmten Funktion zu einem bestimmten Zeitpunkt erfordert. Durch das Erzwingen des Aufrufs können Sie garantieren, dass die gewünschte Funktion aufgerufen wird, anstatt die KI höflich darum bitten zu müssen.
2. **Debugging und Testen:** Bei der Entwicklung und dem Testen von KI-gesteuerten Anwendungen ist die Möglichkeit, Funktionsaufrufe zu erzwingen, für Debugging-Zwecke unschätzbar wertvoll. Durch das explizite Auslösen bestimmter Funktionen können Sie einzelne Komponenten Ihrer Anwendung isolieren und testen. Dies ermöglicht es Ihnen, die Korrektheit der Funktionsimplementierungen zu überprüfen, die Eingangsparameter zu validieren und sicherzustellen, dass die erwarteten Ergebnisse zurückgegeben werden.
3. **Umgang mit Grenzfällen:** Es kann Grenzfälle oder Ausnahmesituationen geben, in denen der dynamische Auswahlprozess der KI eine Funktion möglicherweise nicht auswählt, obwohl Sie aufgrund externer Prozesse wissen, dass sie ausgeführt werden sollte. In solchen Fällen ermöglicht die Fähigkeit, einen Funktionsaufruf zu erzwingen, diese Situationen explizit zu behandeln. Definieren Sie Regeln oder Bedingungen in Ihrer Anwendungslogik, um zu bestimmen, wann das Ermessen der KI überschrieben werden soll.

4. **Konsistenz und Reproduzierbarkeit:** Wenn Sie eine bestimmte Abfolge von Funktionen haben, die in einer bestimmten Reihenfolge ausgeführt werden müssen, garantiert das Erzwingen der Aufrufe, dass dieselbe Sequenz jedes Mal eingehalten wird. Dies ist besonders wichtig in Anwendungen, bei denen Konsistenz und vorhersehbares Verhalten kritisch sind, wie beispielsweise in Finanzsystemen oder wissenschaftlichen Simulationen.
5. **Leistungsoptimierung:** In manchen Fällen kann das Erzwingen eines Funktionsaufrufs zu Leistungsoptimierungen führen. Wenn Sie wissen, dass eine bestimmte Funktion für eine bestimmte Aufgabe erforderlich ist und der dynamische Auswahlprozess der KI möglicherweise unnötigen Overhead verursacht, können Sie den Auswahlprozess umgehen und die erforderliche Funktion direkt aufrufen. Dies kann dazu beitragen, die Latenzzeit zu reduzieren und die Gesamteffizienz Ihrer Anwendung zu verbessern.

Zusammenfassend lässt sich sagen, dass die Möglichkeit, Funktionsaufrufe in KI-gesteuerten Anwendungen zu erzwingen, explizite Kontrolle bietet, beim Debugging und Testen hilft, Grenzfälle handhabt und Konsistenz und Reproduzierbarkeit gewährleistet. Es ist ein leistungsstarkes Werkzeug in Ihrem Arsenal, aber wir müssen noch einen weiteren Aspekt dieser wichtigen Funktion besprechen.



In vielen Entscheidungsfindungsfällen möchten wir, dass das Modell immer einen Funktionsaufruf durchführt und möglicherweise nie nur mit seinem internen Wissen antwortet. Wenn Sie zum Beispiel zwischen mehreren Modellen routing betreiben, die auf verschiedene Aufgaben spezialisiert sind (mehrsprachige Eingabe, Mathematik usw.), verwenden Sie möglicherweise das funktionsaufrufende Modell, um Anfragen an eines der Hilfsmodelle zu delegieren und nie unabhängig zu antworten.

Werkzeugauswahlparameter

GPT-4 und andere Sprachmodelle, die Funktionsaufrufe unterstützen, bieten einen `tool_choice`-Parameter zur Steuerung, ob die Werkzeugverwendung als Teil einer Completion erforderlich ist. Dieser Parameter hat drei mögliche Werte:

- `auto` gibt der KI volle Entscheidungsfreiheit über die Verwendung eines Werkzeugs oder eine einfache Antwort
- `required` teilt der KI mit, dass sie *zwingend* ein Werkzeug aufrufen *muss* anstatt zu antworten, überlässt aber die Auswahl des Werkzeugs der KI
- Die dritte Option besteht darin, den Parameter des `name_of_function` festzulegen, den Sie erzwingen möchten. Mehr dazu im nächsten Abschnitt.



Beachten Sie, dass wenn Sie `tool choice` auf `required` setzen, das Modell gezwungen wird, die relevanteste Funktion aus den bereitgestellten auszuwählen, auch wenn keine wirklich zum Prompt passt. Zum Zeitpunkt der Veröffentlichung ist mir kein Modell bekannt, das eine leere `tool_calls`-Antwort zurückgibt oder auf andere Weise mitteilt, dass es keine geeignete Funktion zum Aufrufen gefunden hat.

Erzwingen einer Funktion für strukturierte Ausgabe

Die Möglichkeit, einen Funktionsaufruf zu erzwingen, bietet Ihnen einen Weg, strukturierte Daten aus einer Chat-Completion zu erhalten, anstatt diese selbst aus der Klartext-Antwort extrahieren zu müssen.

Warum ist das Erzwingen von Funktionen für strukturierte Ausgabe so wichtig? Ganz einfach, weil die Extraktion strukturierter Daten aus LLM-Ausgaben eine mühsame Angelegenheit ist. Sie können sich das Leben etwas erleichtern, indem Sie nach Daten in XML fragen, aber dann müssen Sie XML parsen. Und was machen Sie, wenn dieses XML fehlt, weil Ihre KI antwortet: “Es tut mir leid, aber ich kann die angeforderten Daten nicht generieren, weil bla, bla, bla...”

Bei der Verwendung von Werkzeugen auf diese Weise:

- Sollten Sie wahrscheinlich nur ein einzelnes Werkzeug in Ihrer Anfrage definieren
- Denken Sie daran, die Verwendung seiner Funktion über den `tool_choice`-Parameter zu erzwingen
- Bedenken Sie, dass das Modell die Eingabe an das Werkzeug weiterleitet, daher sollten der Name des Werkzeugs und die Beschreibung aus der Perspektive des Modells erfolgen, nicht aus Ihrer

Dieser letzte Punkt verdient ein Beispiel zur Verdeutlichung. Angenommen, Sie bitten die KI um eine Stimmungsanalyse eines Benutzertextes. Der Name der Funktion wäre nicht `analyze_sentiment`, sondern eher etwas wie `save_sentiment_analysis`. Die KI ist diejenige, die die Stimmungsanalyse durchführt, *nicht das Werkzeug*. Alles, was das Werkzeug tut (aus der Perspektive der KI), ist das Speichern der Analyseergebnisse.

Hier ist ein Beispiel für die Verwendung von Claude 3, um eine Zusammenfassung eines Bildes in gut strukturiertem JSON zu erfassen, diesmal von der Kommandozeile aus mit `curl`:


```

1  curl https://api.anthropic.com/v1/messages \
2      --header "content-type: application/json" \
3      --header "x-api-key: $ANTHROPIC_API_KEY" \
4      --header "anthropic-version: 2023-06-01" \
5      --header "anthropic-beta: tools-2024-04-04" \
6      --data \
7      '{
8          "model": "claude-3-sonnet-20240229",
9          "max_tokens": 1024,
10         "tools": [{
11             "name": "record_summary",
12             "description": "Record summary of image into well-structured JSON.",
13             "input_schema": {
14                 "type": "object",
15                 "properties": {
16                     "key_colors": {
17                         "type": "array",
18                         "items": {
19                             "type": "object",
20                             "properties": {
21                                 "r": {
22                                     "type": "number",
23                                     "description": "red value [0.0, 1.0]"
24                                 },
25                                 "g": {
26                                     "type": "number",
27                                     "description": "green value [0.0, 1.0]"
28                                 },
29                                 "b": {
30                                     "type": "number",
31                                     "description": "blue value [0.0, 1.0]"
32                                 },
33                                 "name": {
34                                     "type": "string",
35                                     "description": "Human-readable color name
36                                         in snake_case, e.g.
37                                         \"olive_green\"or
38                                         \"turquoise\""
39                                 }
40                             },
41                             "required": [ "r", "g", "b", "name" ]
42                         },

```

```

43         "description": "Key colors in the image. Four or less."
44     },
45     "description": {
46         "type": "string",
47         "description": "Image description. 1-2 sentences max."
48     },
49     "estimated_year": {
50         "type": "integer",
51         "description": "Estimated year that the image was taken,
52                         if is it a photo. Only set this if the
53                         image appears to be non-fictional.
54                         Rough estimates are okay!"
55     }
56 },
57 "required": [ "key_colors", "description" ]
58 }
59 ]],
60 "messages": [
61     {
62         "role": "user",
63         "content": [
64             {
65                 "type": "image",
66                 "source": {
67                     "type": "base64",
68                     "media_type": "'$IMAGE_MEDIA_TYPE'",
69                     "data": "'$IMAGE_BASE64'"
70                 }
71             },
72             {
73                 "type": "text",
74                 "text": "Use `record_summary` to describe this image."
75             }
76         ]
77     }
78 ]
79 }'

```

In dem bereitgestellten Beispiel verwenden wir das Claude 3-Modell von Anthropic, um eine strukturierte JSON-Zusammenfassung eines Bildes zu generieren. Hier ist die Funktionsweise:

1. Wir definieren ein einzelnes Tool namens `record_summary` im `tools`-Array der Anfrage-Payload. Dieses Tool ist dafür verantwortlich, eine Zusammenfassung des Bildes in wohlstrukturiertem JSON zu erfassen.
2. Das `record_summary`-Tool verfügt über ein `input_schema`, das die erwartete Struktur der JSON-Ausgabe festlegt. Es definiert drei Eigenschaften:
 - `key_colors`: Ein Array von Objekten, die die wichtigsten Farben im Bild darstellen. Jedes Farbojekt hat Eigenschaften für die Rot-, Grün- und Blau-Werte (von 0,0 bis 1,0) und einen menschenlesbaren Farbnamen im `snake_case`-Format.
 - `description`: Eine String-Eigenschaft für eine kurze Beschreibung des Bildes, begrenzt auf 1-2 Sätze.
 - `estimated_year`: Eine optionale Integer-Eigenschaft für das geschätzte Jahr, in dem das Bild aufgenommen wurde, falls es sich um ein nicht-fiktionales Foto zu handeln scheint.
3. Im `messages`-Array stellen wir die Bilddaten als base64-codierten String zusammen mit dem Medientyp bereit. Dies ermöglicht es dem Modell, das Bild als Teil der Eingabe zu verarbeiten.
4. Wir weisen Claude außerdem an, das `record_summary`-Tool zu verwenden, um das Bild zu beschreiben.
5. Wenn die Anfrage an das Claude 3-Modell gesendet wird, analysiert es das Bild und generiert eine JSON-Zusammenfassung basierend auf dem spezifizierten `input_schema`. Das Modell extrahiert die wichtigsten Farben, liefert eine kurze Beschreibung und schätzt das Jahr der Aufnahme (falls zutreffend).
6. Die generierte JSON-Zusammenfassung wird als Parameter an das `record_summary`-Tool übergeben und bietet eine strukturierte Darstellung der wichtigsten Eigenschaften des Bildes.

Durch die Verwendung des `record_summary`-Tools mit einem klar definierten `input_schema` können wir eine strukturierte JSON-Zusammenfassung eines Bildes

erhalten, ohne auf reine Textextraktion angewiesen zu sein. Dieser Ansatz stellt sicher, dass die Ausgabe einem einheitlichen Format folgt und von nachgelagerten Komponenten der Anwendung einfach geparkt und verarbeitet werden kann.

Die Fähigkeit, einen Funktionsaufruf zu erzwingen und die erwartete Ausgabestruktur zu spezifizieren, ist eine leistungsstarke Funktion der Tool-Nutzung in KI-gesteuerten Anwendungen. Sie ermöglicht es Entwicklern, mehr Kontrolle über die generierte Ausgabe zu haben und vereinfacht die Integration von KI-generierten Daten in den Arbeitsablauf ihrer Anwendung.

Ausführung von Funktion(en)

Sie haben Funktionen definiert und Ihre KI aufgefordert, die daraufhin beschlossen hat, eine Ihrer Funktionen aufzurufen. Jetzt ist es Zeit für Ihren Anwendungscode oder Ihre Bibliothek, wenn Sie ein Ruby-Gem wie [raix-rails](#) verwenden, den Funktionsaufruf und seine Parameter an die entsprechende Implementierung *in Ihrem Anwendungscode* weiterzuleiten.

Ihr Anwendungscode entscheidet, was mit den Ergebnissen der Funktionsausführung geschehen soll. Vielleicht besteht die Aktion aus einer einzigen Codezeile in einem Lambda, oder vielleicht beinhaltet sie den Aufruf einer externen API. Möglicherweise umfasst sie den Aufruf einer anderen KI-Komponente oder sogar Hunderte oder Tausende von Codezeilen im Rest Ihres Systems. Das liegt ganz bei Ihnen.

Manchmal ist der Funktionsaufruf das Ende der Operation, aber wenn die Ergebnisse Informationen in einer Chain of Thought darstellen, die von der KI fortgesetzt werden soll, muss Ihr Anwendungscode die Ausführungsergebnisse in das Chat-Transkript einfügen und die KI mit der Verarbeitung fortfahren lassen.

Hier ist zum Beispiel eine [Raix](#)-Funktionsdeklaration, die von Olympias AccountManager zur Kommunikation mit unseren Kunden als Teil einer Intelligenzen Workflow-Orchestrierung für den Kundenservice verwendet wird.

```
1 class AccountManager
2   include Raix::ChatCompletion
3   include Raix::FunctionDispatch
4
5   # lots of other functions...
6
7   function :notify_account_owner,
8     "Don't share UUID. Mention dollars if subscription changed",
9     message: { type: "string" } do |arguments|
10     account.owner.freeform_notify(
11       subject: "Account Change Notification",
12       message: arguments[:message]
13     )
14     "Notified account owner"
15   end
```

Möglicherweise ist nicht sofort klar, was hier passiert, daher werde ich es aufschlüsseln.

1. Die AccountManager-Klasse definiert viele Funktionen im Zusammenhang mit der Kontoverwaltung. Sie kann Ihren Plan ändern, Teammitglieder hinzufügen und entfernen, und vieles mehr.
2. Die übergeordneten Anweisungen teilen AccountManager mit, dass er den Kontoinhaber über die Ergebnisse der Kontoänderungsanfrage mittels der Funktion `notify_account_owner` benachrichtigen soll.
3. Die präzise Definition der Funktion enthält:

- Name
- Beschreibung
- Parameter `message: { type: "string" }`
- einen Block, der bei Funktionsaufruf ausgeführt wird

Nach der Aktualisierung des Transkripts mit den Ergebnissen des Funktionsblocks wird die `chat_completion`-Methode erneut aufgerufen. Diese Methode ist dafür verantwortlich, das aktualisierte Konversationstranskript zur weiteren

Verarbeitung an das KI-Modell zurückzusenden. Wir bezeichnen diesen Prozess als *Konversationsschleife*.

Wenn das KI-Modell eine neue Chat-Vervollständigungsanfrage mit einem aktualisierten Transkript erhält, hat es Zugriff auf die Ergebnisse der zuvor ausgeführten Funktion. Es kann diese Ergebnisse analysieren, sie in seinen Entscheidungsprozess einbeziehen und die nächste Antwort oder Aktion basierend auf dem kumulativen Kontext der Konversation generieren. Es kann basierend auf dem aktualisierten Kontext weitere Funktionen ausführen oder eine abschließende Antwort auf die ursprüngliche Anfrage generieren, wenn es feststellt, dass keine weiteren Funktionsaufrufe erforderlich sind.

Optionale Fortsetzung der ursprünglichen Anfrage

Wenn Sie die Werkzeugergebnisse zurück an das LLM senden und die Verarbeitung der ursprünglichen Anfrage fortsetzen, verwendet die KI diese Ergebnisse, um entweder zusätzliche Funktionen aufzurufen oder eine endgültige Textantwort zu generieren.



Einige Modelle wie Coheres [Command-R](#) können in ihren Antworten die spezifischen Werkzeuge zitieren, die sie verwendet haben, was zusätzliche Transparenz und Nachverfolgbarkeit bietet.

Je nach verwendetem Modell befinden sich die Ergebnisse des Funktionsaufrufs in Transkriptnachrichten mit einer eigenen speziellen Rolle oder werden in einer anderen Syntax dargestellt. Der wichtige Teil ist jedoch, dass diese Daten im Transkript enthalten sind, damit die KI sie bei der Entscheidung über das weitere Vorgehen berücksichtigen kann.



Ein häufiger (und potenziell teurer) Fehler besteht darin, zu vergessen, die Funktionsergebnisse dem Transkript hinzuzufügen, bevor man fortfährt. Infolgedessen wird die KI im Wesentlichen auf die gleiche Weise aufgefordert wie vor dem ersten Funktionsaufruf. Mit anderen Worten: Aus Sicht der KI hat sie die Funktion noch nicht aufgerufen. Also ruft sie sie wieder auf. Und wieder. Und wieder, bis Sie sie unterbrechen. Hoffentlich war Ihr Kontext nicht zu groß und Ihr Modell nicht zu teuer!

Best Practices für die Werkzeugnutzung

Um das Beste aus der Werkzeugnutzung herauszuholen, beachten Sie die folgenden Best Practices.

Beschreibende Definitionen

Stellen Sie klare und beschreibende Namen und Beschreibungen für jedes Werkzeug und seine Eingangsparameter bereit. Dies hilft dem LLM besser zu verstehen, welchen Zweck und welche Fähigkeiten jedes Werkzeug hat.

Aus Erfahrung kann ich Ihnen sagen, dass die allgemeine Weisheit, die besagt, dass “Benennung schwierig ist”, auch hier zutrifft; ich habe dramatisch unterschiedliche Ergebnisse von LLMs gesehen, nur durch die Änderung von Funktionsnamen oder der Formulierung von Beschreibungen. Manchmal verbessert das Entfernen von Beschreibungen sogar die Leistung.

Verarbeitung von Werkzeugergebnissen

Achten Sie beim Zurücksenden von Werkzeugergebnissen an das LLM darauf, dass diese gut strukturiert und umfassend sind. Verwenden Sie aussagekräftige Schlüssel und Werte, um die Ausgabe jedes Werkzeugs darzustellen. Experimentieren Sie mit verschiedenen Formaten und finden Sie heraus, welches am besten funktioniert, von JSON bis hin zu Klartext.

Der [Ergebnisinterpret](#) geht diese Herausforderung an, indem er KI einsetzt, um die Ergebnisse zu analysieren und benutzerfreundliche Erklärungen, Zusammenfassungen oder wichtige Erkenntnisse bereitzustellen.

Fehlerbehandlung

Implementieren Sie robuste Fehlerbehandlungsmechanismen, um Fälle zu behandeln, in denen das LLM möglicherweise ungültige oder nicht unterstützte Eingangsparameter für Werkzeugaufrufe generiert. Behandeln und beheben Sie alle Fehler, die während der Werkzeugausführung auftreten können, auf elegante Weise.

Eine äußerst angenehme Eigenschaft der KI ist, dass sie Fehlermeldungen versteht! Das bedeutet, wenn Sie in einer schnellen und einfachen Denkweise arbeiten, können Sie einfach alle Ausnahmen abfangen, die bei der Implementierung eines Werkzeugs generiert werden, und sie an die KI zurückgeben, damit sie weiß, was passiert ist!

Hier ist zum Beispiel eine verschlankte Version der Implementierung der Google-Suche in Olympia:


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

Google-Suchen in Olympia sind ein zweistufiger Prozess. Zuerst führen Sie die Suche durch, dann fassen Sie die Ergebnisse zusammen. Bei einem Fehler, egal welcher Art, wird die Fehlermeldung verpackt und an die KI zurückgesendet. Diese Technik ist die Grundlage für praktisch alle *Intelligente Fehlerbehandlung*-Muster.

Nehmen wir zum Beispiel an, dass der GoogleSearch-API-Aufruf aufgrund einer 503 Service-Unavailable-Ausnahme fehlschlägt. Diese wird bis zur obersten Fehlerbehandlung weitergereicht, und die Beschreibung des Fehlers wird als Funktionsergebnis an die KI zurückgesendet. Anstatt dem Benutzer einfach einen leeren Bildschirm oder einen technischen Fehler anzuzeigen, sagt die KI etwa: “Es tut mir leid, aber ich kann derzeit nicht auf meine Google-Such-Funktionen zugreifen. Ich kann es später noch einmal versuchen, wenn Sie möchten.”

Dies mag wie ein cleverer Trick erscheinen, aber betrachten Sie eine andere Art von Fehler, bei dem die KI eine externe API aufruft und direkte Kontrolle über die zu übergebenden Parameter hat. Vielleicht hat sie einen Fehler bei der Generierung dieser Parameter gemacht? Vorausgesetzt, die Fehlermeldung der externen API ist detailliert genug, bedeutet die Rückgabe der Fehlermeldung an die aufrufende KI, dass sie diese Parameter überdenken und es erneut versuchen kann. Automatisch. Egal, was der Fehler war.

Denken Sie nun darüber nach, was es bedeuten würde, diese Art von robuster Fehlerbehandlung in *normalem* Code zu replizieren. Es ist praktisch unmöglich.

Iterative Verfeinerung

Wenn das LLM nicht die geeigneten Werkzeuge empfiehlt oder suboptimale Antworten generiert, iterieren Sie die Werkzeugdefinitionen, Beschreibungen und Eingabeparameter. Verfeinern und verbessern Sie die Werkzeugeinrichtung kontinuierlich basierend auf dem beobachteten Verhalten und den gewünschten Ergebnissen.

1. Beginnen Sie mit einfachen Werkzeugdefinitionen: Fangen Sie an, indem Sie Werkzeuge mit klaren und präzisen Namen, Beschreibungen und Eingabeparametern definieren. Vermeiden Sie es zunächst, die Werkzeugeinrichtung zu verkomplizieren, und konzentrieren Sie sich auf die Kernfunktionalität. Wenn Sie zum Beispiel die Ergebnisse einer Stimmungsanalyse speichern möchten, beginnen Sie mit einer grundlegenden Definition wie:

```
1  {
2    "name": "save_sentiment_score",
3    "description": "Analyze user-provided text and generate sentiment score",
4    "parameters": {
5      "type": "object",
6      "properties": {
7        "score": {
8          "type": "float",
9          "description": "sentiment score from -1 (negative) to 1 (positive)"
10       }
11     },
12     "required": ["score"]
13   }
14 }
```

2. Testen und beobachten: Sobald Sie die ersten Werkzeugdefinitionen erstellt haben, testen Sie diese mit verschiedenen Prompts und beobachten Sie, wie das LLM mit dem Werkzeug interagiert. Achten Sie auf die Qualität und Relevanz der

generierten Antworten. Wenn das LLM suboptimale Antworten generiert, ist es Zeit, die Werkzeugdefinitionen zu verfeinern.

3. Beschreibungen verfeinern: Wenn das LLM den Zweck eines Werkzeugs missversteht, versuchen Sie, die Beschreibung des Werkzeugs zu verfeinern. Stellen Sie mehr Kontext, Beispiele oder Erläuterungen bereit, um das LLM bei der effektiven Nutzung des Werkzeugs zu unterstützen. Sie können zum Beispiel die Beschreibung des Stimmungsanalyse-Werkzeugs aktualisieren, um spezifischer auf den *emotionalen Ton* des zu analysierenden Textes einzugehen:

```
1 {  
2   "name": "save_sentiment_score",  
3   "description": "Determine the overall emotional tone of a piece of text,  
4     such as customer reviews, social media posts, or feedback comments.",  
5   ...  
6 }
```

4. Eingabeparameter anpassen: Wenn das LLM ungültige oder irrelevante Eingabeparameter für ein Werkzeug generiert, sollten Sie die Parameterdefinitionen anpassen. Fügen Sie spezifischere Einschränkungen, Validierungsregeln oder Beispiele hinzu, um das erwartete Eingabeformat zu verdeutlichen.
5. Auf Basis von Feedback iterieren: Überwachen Sie kontinuierlich die Leistung Ihrer Werkzeuge und sammeln Sie Feedback von Benutzern oder Stakeholdern. Nutzen Sie dieses Feedback, um Verbesserungsbereiche zu identifizieren und nehmen Sie iterative Verfeinerungen an den Werkzeugdefinitionen vor. Wenn Benutzer zum Beispiel berichten, dass die Analyse Sarkasmus nicht gut erkennt, können Sie einen entsprechenden Hinweis in die Beschreibung aufnehmen:

```
1 {  
2   "name": "save_sentiment_score",  
3   "description": "Analyze the sentiment of a given text and return a sentiment  
4     score between -1 (negative) and 1 (positive). Note: Sarcasm should be  
5     considered negative.",  
6   ...  
7 }
```

Durch die iterative Verfeinerung Ihrer Werkzeugdefinitionen basierend auf beobachtetem Verhalten und Feedback können Sie die Leistung und Effektivität Ihrer KI-gesteuerten Anwendung schrittweise verbessern. Denken Sie daran, die Werkzeugdefinitionen klar, präzise und auf die spezifische Aufgabe fokussiert zu halten. Testen und validieren Sie die Werkzeuginteraktionen regelmäßig, um sicherzustellen, dass sie mit Ihren gewünschten Ergebnissen übereinstimmen.

Zusammenstellung und Verkettung von Werkzeugen

Einer der leistungsfähigsten Aspekte der Werkzeugnutzung, der bisher nur angedeutet wurde, ist die Möglichkeit, mehrere Werkzeuge zusammenzustellen und zu verketteten, um komplexe Aufgaben zu bewältigen. Durch sorgfältige Gestaltung Ihrer Werkzeugdefinitionen und deren Ein-/Ausgabeformate können Sie wiederverwendbare Bausteine erstellen, die sich auf verschiedene Weise kombinieren lassen.

Betrachten wir ein Beispiel, bei dem Sie eine Datenanalysepipeline für Ihre KI-gesteuerte Anwendung erstellen. Sie könnten die folgenden Werkzeuge haben:

1. **DataRetrieval:** Ein Werkzeug, das Daten aus einer Datenbank oder API basierend auf festgelegten Kriterien abrufen.
2. **DataProcessing:** Ein Werkzeug, das Berechnungen, Transformationen oder Aggregationen der abgerufenen Daten durchführt.

3. **DataVisualization:** Ein Werkzeug, das die verarbeiteten Daten in einem benutzerfreundlichen Format wie Diagrammen oder Grafiken darstellt.

Durch die Verkettung dieser Werkzeuge können Sie einen leistungsfähigen Workflow erstellen, der relevante Daten abrufen, verarbeitet und die Ergebnisse aussagekräftig präsentiert. So könnte der Werkzeugnutzungs-Workflow aussehen:

1. Das LLM erhält eine Benutzeranfrage nach Erkenntnissen über Verkaufsdaten für eine bestimmte Produktkategorie.
2. Das LLM wählt das DataRetrieval-Werkzeug aus und generiert die entsprechenden Eingangsparameter, um die relevanten Verkaufsdaten aus der Datenbank abzurufen.
3. Die abgerufenen Daten werden an das DataProcessing-Werkzeug "weitergegeben", das Metriken wie Gesamtumsatz, durchschnittlichen Verkaufspreis und Wachstumsrate berechnet.
4. Die verarbeiteten Daten werden dann vom DataVisualization-Werkzeug verarbeitet, das ein ansprechendes Diagramm oder eine Grafik erstellt, um die Erkenntnisse darzustellen, wobei die URL des Diagramms an das LLM zurückgegeben wird.
5. Schließlich generiert das LLM eine formatierte Antwort auf die Benutzeranfrage unter Verwendung von Markdown, die die visualisierten Daten einbindet und eine Zusammenfassung der wichtigsten Erkenntnisse liefert.

Durch die Zusammenstellung dieser Werkzeuge können Sie einen nahtlosen Datenanalyse-Workflow erstellen, der sich leicht in Ihre Anwendung integrieren lässt. Die Schönheit dieses Ansatzes liegt darin, dass jedes Werkzeug unabhängig entwickelt und getestet und dann auf verschiedene Weise kombiniert werden kann, um verschiedene Probleme zu lösen.

Um eine reibungslose Zusammenstellung und Verkettung von Werkzeugen zu ermöglichen, ist es wichtig, klare Ein- und Ausgabeformate für jedes Werkzeug zu definieren.

Zum Beispiel könnte das DataRetrieval-Werkzeug Parameter wie Datenbankverbindungsdetails, Tabellename und Abfragebedingungen akzeptieren und das Ergebnis als strukturiertes JSON-Objekt zurückgeben. Das DataProcessing-Werkzeug kann dann dieses JSON-Objekt als Eingabe erwarten und ein transformiertes JSON-Objekt als Ausgabe produzieren. Durch die Standardisierung des Datenflusses zwischen Werkzeugen können Sie Kompatibilität und Wiederverwendbarkeit gewährleisten.

Wenn Sie Ihr Werkzeug-Ökosystem entwerfen, denken Sie darüber nach, wie verschiedene Werkzeuge kombiniert werden können, um häufige Anwendungsfälle in Ihrer Anwendung zu adressieren. Erwägen Sie die Erstellung von High-Level-Werkzeugen, die häufige Workflows oder Geschäftslogik kapseln und es dem LLM erleichtern, diese effektiv auszuwählen und zu nutzen.

Denken Sie daran, dass die Stärke der Werkzeugnutzung in der Flexibilität und Modularität liegt, die sie bietet. Indem Sie komplexe Aufgaben in kleinere, wiederverwendbare Werkzeuge aufteilen, können Sie eine robuste und anpassungsfähige KI-gesteuerte Anwendung erstellen, die eine breite Palette von Herausforderungen bewältigen kann.

Zukünftige Entwicklungen

Mit der Weiterentwicklung des Bereichs der KI-gesteuerten Anwendungsentwicklung können wir weitere Fortschritte bei den Werkzeugnutzungsfähigkeiten erwarten. Einige mögliche zukünftige Richtungen sind:

1. **Mehrstufige Werkzeugnutzung:** LLMs könnten in der Lage sein zu entscheiden, wie oft sie Werkzeuge nutzen müssen, um eine zufriedenstellende Antwort zu generieren. Dies könnte mehrere Runden der Werkzeugauswahl und -ausführung basierend auf Zwischenergebnissen umfassen.

2. **Vordefinierte Werkzeuge:** KI-Plattformen könnten einen Satz vordefinierter Werkzeuge bereitstellen, die Entwickler sofort nutzen können, wie Python-Interpreter, Websuchtools oder gängige Hilfsfunktionen.
3. **Nahtlose Integration:** Mit zunehmender Verbreitung der Werkzeugnutzung können wir eine bessere Integration zwischen KI-Plattformen und beliebten Entwicklungsframeworks erwarten, wodurch es für Entwickler einfacher wird, die Werkzeugnutzung in ihre Anwendungen zu integrieren.

Die Werkzeugnutzung ist eine leistungsfähige Technik, die es Entwicklern ermöglicht, das volle Potenzial von LLMs in KI-gesteuerten Anwendungen zu nutzen. Durch die Verbindung von LLMs mit externen Werkzeugen und Ressourcen können Sie dynamischere, intelligenter und kontextbewusstere Systeme erstellen, die sich an Benutzerbedürfnisse anpassen und wertvolle Erkenntnisse und Aktionen bereitstellen können.

Während die Werkzeugnutzung immense Möglichkeiten bietet, ist es wichtig, sich potenzieller Herausforderungen und Überlegungen bewusst zu sein. Ein wichtiger Aspekt ist die Verwaltung der Komplexität von Werkzeuginteraktionen und die Gewährleistung der Stabilität und Zuverlässigkeit des Gesamtsystems. Sie müssen Szenarien behandeln, in denen Werkzeugaufrufe fehlschlagen, unerwartete Ergebnisse liefern oder Auswirkungen auf die Leistung haben können. Darüber hinaus sollten Sie Sicherheits- und Zugriffskontrollen berücksichtigen, um unbefugte oder böswillige Nutzung von Werkzeugen zu verhindern. Angemessene Fehlerbehandlung, Protokollierung und Überwachungsmechanismen sind entscheidend, um die Integrität und Leistung Ihrer KI-gesteuerten Anwendung zu erhalten.

Während Sie die Möglichkeiten des Werkzeugeinsatzes in Ihren eigenen Projekten erkunden, denken Sie daran, mit klaren Zielsetzungen zu beginnen, gut strukturierte Werkzeugdefinitionen zu erstellen und auf Basis von Feedback und Ergebnissen

zu iterieren. Mit dem richtigen Ansatz und der richtigen Denkweise kann der Werkzeugeinsatz neue Ebenen der Innovation und des Mehrwerts in Ihren KI-gesteuerten Anwendungen erschließen

Stream Processing



Die Übertragung von Streaming-Daten über HTTP, auch bekannt als serverseitige Ereignisse (SSE), ist ein Mechanismus, bei dem der Server kontinuierlich Daten an den Client sendet, sobald diese verfügbar sind, ohne dass der Client diese explizit anfordern muss. Da die Antwort der KI schrittweise generiert wird, ist es sinnvoll, eine reaktionsschnelle Benutzererfahrung zu bieten, indem die Ausgabe der KI bereits während der Generierung angezeigt wird. Tatsächlich bieten alle mir bekannten KI-Provider-APIs Streaming-Antworten als Option in ihren Completion-Endpunkten an.

Der Grund, warum dieses Kapitel hier im Buch, direkt nach [Using Tools](#) erscheint, liegt in der großen Wirkung, die durch die Kombination von Werkzeugen mit Live-KI-Antworten für Benutzer erzielt werden kann. Dies ermöglicht dynamische und interaktive Erfahrungen, bei denen die KI Benutzereingaben verarbeiten, verschiedene Werkzeuge und Funktionen nach eigenem Ermessen nutzen und dann Echtzeitantworten liefern kann.

Um diese nahtlose Interaktion zu erreichen, müssen Sie Stream-Handler schreiben, die sowohl von der KI aufgerufene Werkzeugfunktionen als auch reine Textausgaben an den Endbenutzer weiterleiten können. Die Notwendigkeit, nach der Verarbeitung einer Werkzeugfunktion eine Schleife durchzuführen, fügt der Aufgabe eine interessante Herausforderung hinzu.

Implementierung eines ReplyStream

Um zu demonstrieren, wie Datenstromverarbeitung implementiert werden kann, wird dieses Kapitel eine vereinfachte Version der ReplyStream-Klasse, die in Olympia verwendet wird, eingehend behandeln. Instanzen dieser Klasse können als stream-Parameter in KI-Client-Bibliotheken wie [ruby-openai](#) und [openrouter](#) übergeben werden.

Hier ist ein Beispiel, wie ich ReplyStream in Olympias PromptSubscriber verwende, der über Wisper auf die Erstellung neuer Benutzernachrichten lauscht.

```
1 class PromptSubscriber
2   include Raix::ChatCompletion
3   include Raix::PromptDeclarations
4
5   # many other declarations omitted...
6
7   prompt text: -> { user_message.content },
8   stream: -> { ReplyStream.new(self) },
9   until: -> { bot_message.complete? }
10
11   def message_created(message) # invoked by Wisper
12     return unless message.role.user? && message.content?
13
14     # rest of the implementation omitted...
```

Neben einer context-Referenz auf den Prompt-Subscriber, der sie instanziiert hat, verfügt die ReplyStream-Klasse auch über Instanzvariablen zur Speicherung eines

Puffer für empfangene Daten sowie über Arrays zur Verfolgung von Funktionsnamen und Argumenten, die während der Stream-Verarbeitung aufgerufen werden.

```
1 class ReplyStream
2   attr_accessor :buffer, :f_name, :f_arguments, :context
3
4   delegate :bot_message, :dispatch, to: :context
5
6   def initialize(context)
7     self.context = context
8     self.buffer = []
9     self.f_name = []
10    self.f_arguments = []
11  end
12
13  def call(chunk, bytesize = nil)
14    # ...
15  end
16
17  # ...
18 end
```

Die `initialize`-Methode richtet den Anfangszustand der `ReplyStream`-Instanz ein, wobei der Puffer, der Kontext und andere Variablen initialisiert werden.

Die `call`-Methode ist der Haupteinstiegspunkt für die Verarbeitung der Streaming-Daten. Sie nimmt einen `chunk` von Daten (dargestellt als Hash) und einen optionalen `bytesize`-Parameter entgegen, der in unserem Beispiel nicht verwendet wird. Innerhalb dieser Methode verwendet die Klasse Mustererkennung, um verschiedene Szenarien basierend auf der Struktur des empfangenen Chunks zu behandeln.



Der Aufruf von `deep_symbolize_keys` auf dem Chunk macht die Mustererkennung eleganter, da wir dadurch mit Symbolen statt mit Zeichenketten arbeiten können.

```
1 def call(chunk, _bytesize)
2     case chunk.deep_symbolize_keys
3
4     in { # match function name
5         choices: [
6             {
7                 delta: {
8                     tool_calls: [
9                         { index: index, function: {name: name} }
10                    ]
11                }
12            }
13        ] }
14
15        f_name[index] = name
```

Das erste Muster, nach dem wir suchen, ist ein Werkzeugaufruf zusammen mit seinem zugehörigen Funktionsnamen. Wenn wir einen erkennen, fügen wir ihn in das `f_name`-Array ein. Wir speichern Funktionsnamen in einem indizierten Array, da das Modell zum parallelen Funktionsaufruf fähig ist und mehrere Funktionen gleichzeitig zur Ausführung senden kann.

Paralleler Funktionsaufruf ist die Fähigkeit eines KI-Modells, mehrere Funktionsaufrufe gemeinsam durchzuführen, wodurch die Effekte und Ergebnisse dieser Funktionsaufrufe parallel verarbeitet werden können. Dies ist besonders nützlich, wenn Funktionen lange dauern, und reduziert die Kommunikation mit der API, was wiederum zu erheblichen Einsparungen beim Token-Verbrauch führen kann.

Als Nächstes müssen wir nach den Argumenten suchen, die zu den Funktionsaufrufen gehören.

```

1  in { # match arguments
2    choices: [
3      {
4        delta: {
5          tool_calls: [
6            {
7              index: index, function: {arguments: argument }
8            }
9          ]
10         }
11       }
12     ]}
13
14     f_arguments[index] ||= "" # initialize if not already
15     f_arguments[index] << argument

```

Ähnlich wie bei den Funktionsnamen speichern wir die Argumente in einem indizierten Array.

Als Nächstes achten wir auf normale Benutzernachrichten, die vom Server einzeln als Token ankommen und der Variable `new_content` zugewiesen werden. Wir müssen auch `finish_reason` im Auge behalten. Dieser Wert bleibt `nil`, bis der letzte Teil der Ausgabesequenz erreicht ist.

```

1  in {
2    choices: [
3      { delta: {content: new_content}, finish_reason: finish_reason }
4    ]}
5
6    # you could transmit every chunk to the user here...
7    buffer << new_content.to_s
8
9    if finish_reason.present?
10      finalize
11    elsif new_content.to_s.match?(/\n\n/)
12      send_to_client # ...or buffer and transmit once per paragraph
13    end

```

Wichtig ist, dass wir einen Musterabgleichsausdruck hinzufügen, um Fehlermeldungen zu behandeln, die vom KI-Modellanbieter gesendet werden. In lokalen

Entwicklungsumgebungen werfen wir eine Exception, aber in der Produktion protokollieren wir den Fehler und schließen ab.

```
1  in { error: { message: } }  
2    if Rails.env.local?  
3      raise message  
4    else  
5      Honeybadger.notify("AI Error: #{message}")  
6      finalize  
7    end
```

Der letzte else-Zweig der case-Anweisung wird ausgeführt, wenn keines der vorherigen Muster übereinstimmt. Es ist lediglich eine Absicherung, damit wir erfahren, falls das KI-Modell beginnt, uns unerkannte Chunks zu senden.

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

Die `send_to_client`-Methode ist für das Senden der gepufferten Inhalte an den Client verantwortlich. Sie überprüft, dass der Puffer nicht leer ist, aktualisiert den Inhalt der Bot-Nachricht, rendert die Bot-Nachricht und speichert den Inhalt in der Datenbank, um die Datenpersistenz sicherzustellen.

```

1  def send_to_client
2    # no need to process pure whitespace
3    return if buffer.join.squish.blank?
4
5    # set the buffer content on the bot message
6    content = buffer.join
7    bot_message.content = content
8
9    # save to database so that we never lose data
10   # even if the stream doesn't terminate correctly
11   bot_message.update_column(:content, content)
12
13   # update content via websocket
14   ConversationRenderer.update(bot_message)
15 end

```

Die `finalize`-Methode wird aufgerufen, wenn die Streamverarbeitung abgeschlossen ist. Sie führt die Funktionsaufrufe aus, falls während des Streams welche empfangen wurden, aktualisiert die Bot-Nachricht mit dem endgültigen Inhalt und anderen relevanten Informationen und setzt den Funktionsaufrufsverlauf zurück.

```

1  def finalize
2    if f_name.any?
3      f_name.each_with_index do |name, index|
4        # takes care of calling the function wherever it's implemented
5        dispatch(name:, arguments: JSON.parse(f_arguments[index]))
6      end
7
8      # reset the function call history
9      f_name.clear
10     f_arguments.clear
11   else
12     content = buffer.join.presence
13     bot_message.update!(content:, complete: true)
14     ConversationRenderer.update(bot_message)
15   end
16 end

```

Wenn das Modell beschließt, eine Funktion aufzurufen, müssen Sie diesen Funktionsaufruf (Name und Argumente) so “weiterleiten”, dass er ausgeführt wird und

die `function_call`- und `function_result`-Nachrichten zum Gesprächsprotokoll hinzugefügt werden.

Meiner Erfahrung nach ist es besser, die Erstellung von Funktionsnachrichten an einer zentralen Stelle in Ihrer Codebasis zu behandeln, anstatt sich auf die Werkzeugimplementierungen zu verlassen. Das ist nicht nur übersichtlicher, sondern hat auch einen sehr wichtigen praktischen Grund: Wenn das KI-Modell eine Funktion aufruft und beim erneuten Durchlauf keine entsprechenden Aufrufs- und Ergebnismeldungen im Protokoll sieht, *wird es dieselbe Funktion erneut aufrufen*. Möglicherweise endlos. Denken Sie daran, dass die KI vollständig zustandslos ist - wenn Sie diese Funktionsaufrufe also nicht an sie zurückgeben, sind sie für sie nie passiert.

```

1  # PromptSubscriber#dispatch
2
3  def dispatch(name:, arguments:):
4      # adds a function_call message to the conversation transcript
5      # plus dispatches to tool and returns result
6      conversation.function_call!(name, arguments).then do |result|
7          # add function result message to the transcript
8          conversation.function_result!(name, result)
9      end
10 end

```



Das Löschen des Funktionsaufrufverlaufs nach dem Ausführen ist genauso wichtig wie sicherzustellen, dass der Aufruf und die Ergebnisse in Ihrem Protokoll landen, damit Sie nicht einfach immer wieder dieselben Funktionen bei jeder Schleifenausführung aufrufen.

Die “Konversationsschleife”

Ich erwähne immer wieder Schleifen, aber wenn Sie neu im Bereich Funktionsaufrufe sind, ist möglicherweise nicht offensichtlich, *warum* wir eine Schleife benötigen. Der

Grund ist, dass sobald die KI Sie auffordert, Werkzeugfunktionen in ihrem Namen auszuführen, sie aufhört zu antworten. Es liegt dann an Ihnen, diese Funktionen auszuführen, die Ergebnisse zu sammeln, sie dem Protokoll hinzuzufügen und dann den ursprünglichen Prompt erneut einzureichen, um einen neuen Satz von Funktionsaufrufen oder benutzerorientierten Ergebnissen zu erhalten.

In der `PromptSubscriber`-Klasse verwenden wir die `prompt`-Methode aus dem `PromptDeclarations`-Modul, um das Verhalten der Konversationsschleife zu definieren. Der `until`-Parameter ist auf `-> { bot_message.complete? }` gesetzt, was bedeutet, dass die Schleife so lange fortgesetzt wird, bis die `bot_message` als vollständig markiert ist.

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



Aber wann wird `bot_message` als abgeschlossen markiert? Falls Sie es vergessen haben, schauen Sie noch einmal in Zeile 13 der `finalize`-Methode nach.

Lassen Sie uns die gesamte Streamverarbeitungslogik überprüfen.

1. Der `PromptSubscriber` empfängt eine neue Benutzernachricht über die `message_created`-Methode, die durch das Wisper Pub/Sub-System jedes Mal aufgerufen wird, wenn der Endbenutzer einen neuen Prompt erstellt.
2. Die `prompt`-Klassenmethode definiert deklarativ das Verhalten der Chat-Vervollständigungslogik für den `PromptSubscriber`. Das KI-Modell führt eine Chat-Vervollständigung mit dem Nachrichteninhalt des Benutzers aus, einer neuen Instanz von `ReplyStream` als Stream-Parameter und der angegebenen Schleifenbedingung.

3. Das KI-Modell verarbeitet den Prompt und beginnt mit der Generierung einer Antwort. Während die Antwort gestreamt wird, wird die `call`-Methode der `ReplyStream`-Instanz für jeden Datenblock aufgerufen.
4. Wenn das KI-Modell beschließt, eine Werkzeugfunktion aufzurufen, werden der Funktionsname und die Argumente aus dem Block extrahiert und jeweils in den Arrays `f_name` und `f_arguments` gespeichert.
5. Wenn das KI-Modell benutzersichtbare Inhalte generiert, werden diese gepuffert und über die `send_to_client`-Methode an den Client gesendet.
6. Sobald die Streamverarbeitung abgeschlossen ist, wird die `finalize`-Methode aufgerufen. Wenn während des Streams Werkzeugfunktionen aufgerufen wurden, werden diese über die `dispatch`-Methode des `PromptSubscriber` ausgeführt.
7. Die `dispatch`-Methode fügt dem Gesprächsprotokoll eine `function_call`-Nachricht hinzu, führt die entsprechende Werkzeugfunktion aus und fügt dem Protokoll eine `function_result`-Nachricht mit dem Ergebnis des Funktionsaufrufs hinzu.
8. Nach dem Ausführen der Werkzeugfunktionen wird der Funktionsaufrufverlauf gelöscht, um doppelte Funktionsaufrufe in nachfolgenden Durchläufen zu verhindern.
9. Wenn keine Werkzeugfunktionen aufgerufen wurden, aktualisiert die `finalize`-Methode die `bot_message` mit dem endgültigen Inhalt, markiert sie als abgeschlossen und sendet die aktualisierte Nachricht an den Client.
10. Die Schleifenbedingung `-> { bot_message.complete? }` wird ausgewertet. Wenn die `bot_message` nicht als abgeschlossen markiert ist, wird die Schleife fortgesetzt und der ursprüngliche Prompt wird mit dem aktualisierten Gesprächsprotokoll erneut eingereicht.
11. Die Schritte 3-10 werden wiederholt, bis die `bot_message` als abgeschlossen markiert ist, was anzeigt, dass das KI-Modell seine Antwort fertig generiert hat und keine weiteren Werkzeugfunktionen ausgeführt werden müssen.

Durch die Implementierung dieser Konversationsschleife ermöglichen Sie dem KI-Modell, in eine Hin-und-her-Interaktion mit der Anwendung zu treten, bei der es nach Bedarf Werkzeugfunktionen ausführt und benutzersichtbare Antworten generiert, bis das Gespräch zu einem natürlichen Abschluss kommt.

Die Kombination aus Streamverarbeitung und Konversationsschleife ermöglicht dynamische und interaktive KI-gestützte Erlebnisse, bei denen das KI-Modell Benutzereingaben verarbeiten, verschiedene Werkzeuge und Funktionen nutzen und Echtzeitanworten basierend auf dem sich entwickelnden Gesprächskontext bereitstellen kann.

Automatische Fortsetzung

Es ist wichtig, sich der Ausgabebeschränkungen der KI bewusst zu sein. Die meisten Modelle haben eine maximale Anzahl von Tokens, die sie in einer einzelnen Antwort generieren können, die durch den Parameter `max_tokens` festgelegt wird. Wenn das KI-Modell diese Grenze während der Generierung einer Antwort erreicht, wird es abrupt stoppen und anzeigen, dass die Ausgabe abgeschnitten wurde.

In der Streaming-Antwort der KI-Plattform-API können Sie diese Situation erkennen, indem Sie die Variable `finish_reason` im Block überprüfen. Wenn die `finish_reason` auf `"length"` gesetzt ist (oder einen anderen modellspezifischen Schlüsselwert), bedeutet dies, dass das Modell während der Generierung sein maximales Token-Limit erreicht hat und die Ausgabe vorzeitig beendet wurde.

Eine Möglichkeit, dieses Szenario elegant zu handhaben und eine nahtlose Benutzererfahrung zu bieten, ist die Implementierung eines automatischen Fortsetzungsmechanismus in Ihrer Streamverarbeitungslogik. Durch das Hinzufügen einer Mustererkennung für längenbezogene Beendigungsgründe können Sie wählen, die Schleife fortzusetzen und die Ausgabe automatisch dort fortzusetzen, wo sie unterbrochen wurde.

Hier ist ein absichtlich vereinfachtes Beispiel, wie Sie die `call`-Methode in der `ReplyStream`-Klasse modifizieren können, um die automatische Fortsetzung zu unterstützen:

```

1  LENGTH_STOPS = %w[length MAX_TOKENS]
2
3  def call(chunk, _bytesize)
4    case chunk.deep_symbolize_keys
5      # ...
6
7      in {
8        choices: [
9          { delta: {content: new_content},
10            finish_reason: finish_reason } ] }
11
12      buffer << new_content.to_s
13
14      if finish_reason.blank?
15        send_to_client if new_content.to_s.match?(/\n\n/)
16      elsif LENGTH_STOPS.include?(finish_reason)
17        continue_cutoff
18      else
19        finalize
20      end
21
22      # ...
23    end
24  end
25
26  private
27
28  def continue_cutoff
29    conversation.bot_message!(buffer.join, visible: false)
30    conversation.user_message!("please continue", visible: false)
31    bot_message.update_column(:created_at, Time.current)
32  end

```

In dieser modifizierten Version fügen wir, wenn der `finish_reason` auf abgeschnittene Ausgabe hinweist, anstatt den Stream zu beenden, ein Nachrichtenpaar zum Transkript hinzu, ohne es zu finalisieren, verschieben die ursprüngliche benutzergerichtete

Antwortnachricht durch Aktualisierung ihres `created_at`-Attributs an das “Ende” des Transkripts und lassen dann die Schleife weiterlaufen, sodass die KI dort weitergeneriert, wo sie aufgehört hat.

Denken Sie daran, dass der KI-Completion-Endpunkt zustandslos ist. Er “kennt” nur das, was Sie ihm über das Transkript mitteilen. In diesem Fall kommunizieren wir der KI, dass sie unterbrochen wurde, indem wir (für den Endbenutzer) “unsichtbare” Nachrichten zum Transkript hinzufügen. Beachten Sie jedoch, dass dies ein bewusst vereinfachtes Beispiel ist. Eine echte Implementierung müsste ein weitergehendes Transkript-Management durchführen, um sicherzustellen, dass wir keine Token verschwenden und/oder die KI nicht durch duplizierte Assistenten-Nachrichten im Transkript verwirren.

Eine echte Implementierung der automatischen Fortsetzung sollte auch eine sogenannte “Circuit-Breaker-Logik” enthalten, um unkontrollierte Schleifen zu verhindern. Der Grund dafür ist, dass die KI bei bestimmten Arten von Benutzeranfragen und niedrigen `max_tokens`-Einstellungen endlos benutzgergerichtete Ausgaben in Schleifen erzeugen könnte.

Bedenken Sie, dass jede Schleife eine separate Anfrage erfordert und jede Anfrage Ihr gesamtes Transkript erneut verbraucht. Sie sollten definitiv die Vor- und Nachteile zwischen Benutzererfahrung und API-Nutzung abwägen, wenn Sie entscheiden, ob Sie die automatische Fortsetzung in Ihrer Anwendung implementieren möchten. Automatische Fortsetzung kann besonders bei der Verwendung von Premium-Geschäftsmodellen gefährlich teuer werden.

Fazit

Stream-Verarbeitung ist ein entscheidender Aspekt beim Aufbau von KI-gesteuerten Anwendungen, die Werkzeugnutzung mit Live-KI-Antworten kombinieren. Durch effiziente Handhabung der Streaming-Daten von KI-Plattform-APIs können Sie eine nahtlose und interaktive Benutzererfahrung bieten, große Antworten verarbeiten, die Ressourcennutzung optimieren und Fehler elegant behandeln.

Die bereitgestellte `Conversation::ReplyStream`-Klasse demonstriert, wie Stream-Verarbeitung in einer Ruby-Anwendung mithilfe von Pattern Matching und ereignisgesteuerter Architektur implementiert werden kann. Durch das Verständnis und die Nutzung von Stream-Verarbeitungstechniken können Sie das volle Potenzial der KI-Integration in Ihren Anwendungen erschließen und leistungsstarke sowie fesselnde Benutzererlebnisse bereitstellen.

Selbstheilende Daten



Selbstheilende Daten ist ein leistungsfähiger Ansatz zur Sicherstellung von Datenintegrität, -konsistenz und -qualität in Anwendungen durch die Nutzung der Fähigkeiten von Large Language Models (LLMs). Diese Kategorie von Mustern konzentriert sich auf die Idee, KI zu nutzen, um automatisch Datenanomalien, Inkonsistenzen oder Fehler zu erkennen, zu diagnostizieren und zu korrigieren, wodurch die Belastung für Entwickler reduziert und ein hohes Maß an Datenzuverlässigkeit aufrechterhalten wird.

Im Kern erkennen die Muster für selbstheilende Daten an, dass Daten die Lebensader jeder Anwendung sind und die Sicherstellung ihrer Genauigkeit und Integrität entscheidend für das ordnungsgemäße Funktionieren und die Benutzererfahrung der Anwendung ist. Die Verwaltung und Aufrechterhaltung der Datenqualität kann jedoch eine komplexe und zeitaufwändige Aufgabe sein, besonders wenn Anwendungen in Größe und Komplexität wachsen. Hier kommt die Kraft der KI ins Spiel.

Bei den Mustern für selbstheilende Daten werden KI-Worker eingesetzt, um die Daten Ihrer Anwendung kontinuierlich zu überwachen und zu analysieren. Diese Modelle haben die Fähigkeit, Muster, Beziehungen und Anomalien innerhalb der Daten zu verstehen und zu interpretieren. Durch die Nutzung ihrer Fähigkeiten zur Verarbeitung und zum Verständnis natürlicher Sprache können sie potenzielle Probleme oder Inkonsistenzen in den Daten identifizieren und geeignete Maßnahmen zu deren Behebung ergreifen.

Der Prozess der selbstheilenden Daten umfasst typischerweise mehrere Schlüsselschritte:

1. **Datenüberwachung:** KI-Worker überwachen ständig die Datenströme, Datenbanken oder Speichersysteme der Anwendung und suchen nach Anzeichen von Anomalien, Inkonsistenzen oder Fehlern. Alternativ können Sie eine KI-Komponente als Reaktion auf eine Ausnahme aktivieren.
2. **Anomalieerkennung:** Wenn ein Problem erkannt wird, analysiert der KI-Worker die Daten im Detail, um die spezifische Art und den Umfang des Problems zu identifizieren. Dies könnte die Erkennung fehlender Werte, inkonsistenter Formate oder Daten umfassen, die vordefinierte Regeln oder Einschränkungen verletzen.
3. **Diagnose und Korrektur:** Sobald das Problem identifiziert ist, nutzt der KI-Worker sein Wissen und Verständnis der Datendomäne, um den geeigneten Handlungsablauf zu bestimmen. Dies könnte die automatische Korrektur der Daten, das Ausfüllen fehlender Werte oder das Markieren des Problems für menschliches Eingreifen umfassen, falls erforderlich.
4. **Kontinuierliches Lernen (optional, je nach Anwendungsfall):** Während Ihr KI-Worker verschiedene Datenprobleme erkennt und löst, kann er Ausgaben erstellen, die beschreiben, was passiert ist und wie er darauf reagiert hat. Diese Metadaten können in Lernprozesse eingespeist werden, die es Ihnen (und möglicherweise dem zugrundeliegenden Modell durch Fine-tuning) ermöglichen,

im Laufe der Zeit effektiver und effizienter bei der Identifizierung und Lösung von Datenanomalien zu werden.

Durch die automatische Erkennung und Korrektur von Datenproblemen können Sie sicherstellen, dass Ihre Anwendung mit hochwertigen, zuverlässigen Daten arbeitet. Dies reduziert das Risiko, dass Fehler, Inkonsistenzen oder datenbezogene Bugs die Funktionalität oder Benutzererfahrung der Anwendung beeinträchtigen.

Sobald Sie KI-Worker haben, die sich um die Aufgabe der Datenüberwachung und -korrektur kümmern, können Sie Ihre Bemühungen auf andere kritische Aspekte der Anwendung konzentrieren. Dies spart Zeit und Ressourcen, die sonst für manuelle Datenbereinigung und -wartung aufgewendet würden. Tatsächlich wird die manuelle Verwaltung der Datenqualität mit zunehmender Größe und Komplexität Ihrer Anwendungen immer schwieriger. Die Muster für "Selbstheilende Daten" skalieren effektiv, indem sie die Kraft der KI nutzen, um große Datenmengen zu verarbeiten und Probleme in Echtzeit zu erkennen.



Aufgrund ihrer Natur können sich KI-Modelle mit wenig oder gar keiner Überwachung an sich ändernde Datenmuster, Schemata oder Anforderungen anpassen. Solange ihre Anweisungen ausreichende Orientierung bieten, insbesondere hinsichtlich der beabsichtigten Ergebnisse, kann sich Ihre Anwendung möglicherweise weiterentwickeln und neue Datenszenarien bewältigen, ohne dass umfangreiche manuelle Eingriffe oder Codeänderungen erforderlich sind.

Die Muster für selbstheilende Daten passen gut zu den anderen Kategorien von Mustern, die wir besprochen haben, wie zum Beispiel "Multitude of Workers". Die Fähigkeit zur Selbstheilung von Daten kann als eine spezialisierte Art von Worker betrachtet werden, der sich speziell auf die Sicherstellung von Datenqualität und -integrität konzentriert. Diese Art von Worker arbeitet neben anderen KI-Workern, wobei jeder zu verschiedenen Aspekten der Anwendungsfunktionalität beiträgt.

Die praktische Implementierung von Mustern für selbstheilende Daten erfordert ein sorgfältiges Design und die Integration von KI-Modellen in die Anwendungsarchitektur. Aufgrund der Risiken von Datenverlust und -korruption sollten Sie klare Richtlinien für die Verwendung dieser Technik definieren. Sie sollten auch Faktoren wie Leistung, Skalierbarkeit und Datensicherheit berücksichtigen.

Praktische Fallstudie: Reparatur von fehlerhaftem JSON

Eine der praktischsten und bequemsten Möglichkeiten, selbstheilende Daten zu nutzen, ist auch sehr einfach zu erklären: die Reparatur von fehlerhaftem JSON.

Diese Technik kann auf die häufige Herausforderung angewendet werden, mit unvollkommenen oder inkonsistenten Daten umzugehen, die von LLMs generiert werden, wie zum Beispiel fehlerhaftes JSON, und bietet einen Ansatz zur automatischen Erkennung und Korrektur dieser Probleme.

Bei Olympia begegne ich regelmäßig Szenarien, in denen LLMs JSON-Daten generieren, die nicht vollständig valide sind. Dies kann aus verschiedenen Gründen geschehen, etwa wenn das LLM Kommentare vor oder nach dem eigentlichen JSON-Code hinzufügt oder Syntaxfehler wie fehlende Kommata oder nicht maskierte doppelte Anführungszeichen einführt. Diese Probleme können zu Parse-Fehlern führen und Störungen in der Funktionalität der Anwendung verursachen.

Um dieses Problem zu lösen, habe ich eine praktische Lösung in Form einer JsonFixer-Klasse implementiert. Diese Klasse verkörpert das “Self-Healing Data” Pattern, indem sie das fehlerhafte JSON als Eingabe nimmt und ein LLM nutzt, um es zu reparieren, während dabei so viel Information und Absicht wie möglich erhalten bleibt.

```

1  class JsonFixer
2    include Raix::ChatCompletion
3
4    def call(bad_json, error_message)
5      raise "No data provided" if bad_json.blank? || error_message.blank?
6
7      transcript << {
8        system: "Consider user-provided JSON that generated a parse
9                exception. Do your best to fix it while preserving the
10               original content and intent as much as possible." }
11      transcript << { user: bad_json }
12      transcript << { assistant: "What is the error message?" }
13      transcript << { user: error_message }
14      transcript << { assistant: "Here is the corrected JSON\n```json\n" }
15
16      self.stop = ["````"]
17
18      chat_completion(json: true)
19    end
20
21    def model
22      "mistralai/mixtral-8x7b-instruct:nitro"
23    end
24  end

```



Beachten Sie, wie JsonFixer [Ventriloquist](#) verwendet, um die Antworten der KI zu steuern.

Der Prozess der selbstheilenden JSON-Daten funktioniert wie folgt:

1. **JSON-Generierung:** Ein LLM wird verwendet, um JSON-Daten basierend auf bestimmten Prompts oder Anforderungen zu generieren. Aufgrund der Beschaffenheit von LLMs ist das generierte JSON jedoch nicht immer vollständig gültig. Der JSON-Parser wird natürlich einen `ParserError` ausgeben, wenn man ihm ungültiges JSON übergibt.

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

Beachten Sie, dass die Fehlermeldung auch an den `JSONFixer`-Aufruf übergeben wird, sodass dieser nicht vollständig vermuten muss, was mit den Daten nicht stimmt, insbesondere da der Parser oft genau angibt, wo der Fehler liegt.

2. **LLM-basierte Korrektur:** Die `JSONFixer`-Klasse sendet das fehlerhafte JSON zurück an ein LLM, zusammen mit einer spezifischen Aufforderung oder Anweisung, das JSON zu korrigieren und dabei die ursprünglichen Informationen und Absichten so weit wie möglich zu bewahren. Das LLM, das auf großen Datenmengen trainiert wurde und JSON-Syntax versteht, versucht, die Fehler zu korrigieren und einen gültigen JSON-String zu generieren. [Response Fencing](#) wird verwendet, um die Ausgabe des LLM einzuschränken, und wir wählen Mixtral 8x7B als KI-Modell, da es für diese Art von Aufgabe besonders gut geeignet ist.
3. **Validierung und Integration:** Der vom LLM zurückgegebene korrigierte JSON-String wird von der `JSONFixer`-Klasse selbst geparkt, da sie `chat_completion(json: true)` aufruft. Wenn das korrigierte JSON die Validierung besteht, wird es wieder in den Arbeitsablauf der Anwendung integriert, wodurch die Anwendung die Datenverarbeitung nahtlos fortsetzen kann. Das fehlerhafte JSON wurde “geheilt”.

Obwohl ich meine eigene `JSONFixer`-Implementierung mehrmals geschrieben und überarbeitet habe, bezweifle ich, dass die Gesamtzeit, die in alle diese Versionen investiert wurde, mehr als ein oder zwei Stunden beträgt.

Beachten Sie, dass die Bewahrung der Absicht ein Schlüsselement jedes selbstheilenden Datenmusters ist. Der LLM-basierte Korrekturprozess zielt darauf ab, die ursprünglichen Informationen und Absichten des generierten JSONs so weit wie

möglich zu bewahren. Dies stellt sicher, dass das korrigierte JSON seine semantische Bedeutung beibehält und effektiv im Kontext der Anwendung verwendet werden kann.

Diese praktische Implementierung des “Selbstheilende Daten”-Ansatzes in Olympia zeigt deutlich, wie KI, insbesondere LLMs, zur Lösung realer Datenherausforderungen eingesetzt werden können. Sie demonstriert die Leistungsfähigkeit der Kombination traditioneller Programmier Techniken mit KI-Fähigkeiten zum Aufbau robuster und effizienter Anwendungen.

Postels Gesetz und das “Selbstheilende Daten”-Muster

“Selbstheilende Daten”, wie sie durch die JSONFixer-Klasse veranschaulicht werden, stimmen gut mit dem als Postels Gesetz bekannten Prinzip überein, das auch als Robustheitsprinzip bezeichnet wird. Postels Gesetz besagt:

“Sei konservativ in dem, was du tust, sei liberal in dem, was du von anderen akzeptierst.”

Dieses Prinzip, ursprünglich von Jon Postel, einem Pionier des frühen Internets, formuliert, betont die Wichtigkeit, Systeme zu entwickeln, die tolerant gegenüber verschiedenen oder sogar leicht fehlerhaften Eingaben sind, während sie beim Senden von Ausgaben strikt die spezifizierten Protokolle einhalten.

Im Kontext von “Selbstheilenden Daten” verkörpert die JSONFixer-Klasse Postels Gesetz, indem sie liberal in der Akzeptanz von fehlerhaften oder unvollkommenen JSON-Daten ist, die von LLMs generiert wurden. Sie lehnt nicht sofort ab oder schlägt fehl, wenn sie auf JSON stößt, das nicht strikt dem erwarteten Format entspricht. Stattdessen verfolgt sie einen toleranten Ansatz und versucht, das JSON mithilfe der Leistungsfähigkeit von LLMs zu korrigieren.

Durch die liberale Akzeptanz von unvollkommenem JSON demonstriert die

JSONFixer-Klasse Robustheit und Flexibilität. Sie erkennt an, dass Daten in der realen Welt oft in verschiedenen Formen vorliegen und nicht immer strengen Spezifikationen entsprechen. Durch die geschickte Handhabung und Korrektur dieser Abweichungen stellt die Klasse sicher, dass die Anwendung auch bei unvollkommenen Daten reibungslos funktionieren kann.

Andererseits hält sich die JSONFixer-Klasse auch an den konservativen Aspekt von Postels Gesetz, wenn es um die Ausgabe geht. Nach der Korrektur des JSONs mithilfe von LLMs validiert die Klasse das korrigierte JSON, um sicherzustellen, dass es strikt dem erwarteten Format entspricht. Sie wahrt die Integrität und Korrektheit der Daten, bevor sie sie an andere Teile der Anwendung weitergibt. Dieser konservative Ansatz garantiert, dass die Ausgabe der JSONFixer-Klasse zuverlässig und konsistent ist und fördert die Interoperabilität und verhindert die Ausbreitung von Fehlern.

Interessante Fakten über Jon Postel:

- Jon Postel (1943-1998) war ein amerikanischer Informatiker, der eine entscheidende Rolle bei der Entwicklung des Internets spielte. Er war als "Gott des Internets" für seine bedeutenden Beiträge zu den grundlegenden Protokollen und Standards bekannt.
- Postel war der Herausgeber der Request for Comments (RFC)-Dokumentreihe, einer Serie technischer und organisatorischer Notizen über das Internet. Er verfasste oder war Mitverfasser von über 200 RFCs, einschließlich der grundlegenden Protokolle wie TCP, IP und SMTP.
- Neben seinen technischen Beiträgen war Postel für seinen bescheidenen und kollaborativen Ansatz bekannt. Er glaubte an die Bedeutung der Konsensfindung und der Zusammenarbeit beim Aufbau eines robusten und interoperablen Netzwerks.
- Postel war von 1977 bis zu seinem vorzeitigen Tod 1998 Direktor der Computer Networks Division am Information Sciences Institute (ISI) der University of Southern California (USC).
- In Anerkennung seiner immensen Beiträge wurde Postel 1998 posthum der

prestigeträchtige Turing Award verliehen, der oft als der “Nobelpreis der Informatik” bezeichnet wird.

Die JSONFixer-Klasse fördert Robustheit, Flexibilität und Interoperabilität - Kernwerte, die Postel während seiner gesamten Karriere hochhielt. Indem wir Systeme aufbauen, die tolerant gegenüber Unvollkommenheiten sind und gleichzeitig strikt Protokolle einhalten, können wir Anwendungen erstellen, die angesichts realer Herausforderungen widerstandsfähiger und anpassungsfähiger sind.

Überlegungen und Gegenanzeigen

Die Anwendbarkeit von selbstheilenden Datenansätzen hängt vollständig von der Art der Daten ab, die Ihre Anwendung verarbeitet. Es gibt einen Grund, warum Sie möglicherweise nicht einfach `JSON.parse` per Monkeypatch so modifizieren sollten, dass automatisch *alle JSON-Parsing-Fehler* in Ihrer Anwendung korrigiert werden: Nicht alle Fehler können oder sollten automatisch korrigiert werden.

Selbstheilung ist besonders heikel, wenn sie mit regulatorischen oder Compliance-Anforderungen in Bezug auf Datenhandhabung und -verarbeitung gekoppelt ist. Einige Branchen, wie das Gesundheitswesen und das Finanzwesen, haben so strenge Vorschriften bezüglich Datenintegrität und Nachvollziehbarkeit, dass jegliche Art von “Black Box”-Datenkorrektur ohne angemessene Überwachung oder Protokollierung gegen diese Vorschriften verstoßen könnte. Es ist entscheidend sicherzustellen, dass alle selbstheilenden Datentechniken, die Sie entwickeln, mit den geltenden rechtlichen und regulatorischen Rahmenbedingungen übereinstimmen.

Die Anwendung von selbstheilenden Datentechniken, insbesondere solcher mit KI-Modellen, kann auch erhebliche Auswirkungen auf die Anwendungsleistung und

Ressourcennutzung haben. Die Verarbeitung großer Datenmengen durch KI-Modelle zur Fehlererkennung und -korrektur kann rechnerisch intensiv sein. Es ist wichtig, die Abwägungen zwischen den Vorteilen selbstheilender Daten und den damit verbundenen Leistungs- und Ressourcenkosten zu bewerten.

Lassen Sie uns nun die Faktoren untersuchen, die bei der Entscheidung eine Rolle spielen, wann und wo dieser leistungsstarke Ansatz anzuwenden ist.

Datenkritikalität

Bei der Erwägung der Anwendung von selbstheilenden Datentechniken ist es entscheidend, die Kritikalität der zu verarbeitenden Daten zu bewerten. Die Kritikalitätsstufe bezieht sich auf die Wichtigkeit und Sensibilität der Daten im Kontext Ihrer Anwendung und ihrer Geschäftsdomäne.

In manchen Fällen ist die automatische Korrektur von Datenfehlern möglicherweise nicht angemessen, besonders wenn die Daten hochsensibel sind oder rechtliche Auswirkungen haben. Betrachten Sie beispielsweise die folgenden Szenarien:

1. **Finanztransaktionen:** In Finanzanwendungen wie Banksystemen oder Handelsplattformen ist die Datengenauigkeit von höchster Bedeutung. Selbst kleine Fehler in Finanzdaten können erhebliche Konsequenzen haben, wie falsche Kontoständen, fehlgeleitete Gelder oder fehlerhafte Handelsentscheidungen. In diesen Fällen können automatisierte Korrekturen ohne gründliche Überprüfung und Prüfung inakzeptable Risiken bergen.
2. **Medizinische Aufzeichnungen:** Gesundheitsanwendungen arbeiten mit hochsensiblen und vertraulichen Patientendaten. Ungenauigkeiten in medizinischen Aufzeichnungen können schwerwiegende Auswirkungen auf die Patientensicherheit und Behandlungsentscheidungen haben. Die automatische Änderung medizinischer Daten ohne angemessene Überwachung und Validierung durch qualifiziertes medizinisches Fachpersonal könnte gegen

regulatorische Anforderungen verstoßen und das Wohlergehen der Patienten gefährden.

3. **Rechtsdokumente:** Anwendungen, die rechtliche Dokumente wie Verträge, Vereinbarungen oder Gerichtsakten verarbeiten, erfordern strenge Genauigkeit und Integrität. Selbst kleine Fehler in rechtlichen Daten können erhebliche rechtliche Konsequenzen haben. Automatisierte Korrekturen sind in diesem Bereich möglicherweise nicht angemessen, da die Daten oft eine manuelle Überprüfung und Verifizierung durch Rechtsexperten erfordern, um ihre Gültigkeit und Durchsetzbarkeit sicherzustellen.

In diesen kritischen Datenszenarien überwiegen die Risiken automatisierter Korrekturen oft die potenziellen Vorteile. Die Folgen der Einführung von Fehlern oder falscher Datenmodifikation können schwerwiegend sein und zu finanziellen Verlusten, rechtlichen Haftungen oder sogar Schäden für Einzelpersonen führen.

Bei hochkritischen Daten ist es wichtig, manuelle Überprüfungs- und Validierungsprozesse zu priorisieren. Menschliche Überwachung und Expertise sind entscheidend für die Sicherstellung der Genauigkeit und Integrität der Daten. Automatisierte selbstheilende Techniken können zwar weiterhin eingesetzt werden, um potenzielle Fehler oder Inkonsistenzen zu kennzeichnen, aber die endgültige Entscheidung über Korrekturen sollte menschliches Urteilsvermögen und Genehmigung einbeziehen.

Es ist jedoch wichtig zu beachten, dass nicht alle Daten in einer Anwendung die gleiche Kritikalitätsstufe haben müssen. Innerhalb derselben Anwendung kann es Teilmengen von Daten geben, die weniger sensibel sind oder bei denen Fehler geringere Auswirkungen haben. In solchen Fällen können selbstheilende Datentechniken selektiv auf diese spezifischen Datenteilmengen angewendet werden, während kritische Daten weiterhin manueller Überprüfung unterliegen.

Der Schlüssel liegt darin, die Kritikalität jeder Datenkategorie in Ihrer Anwendung sorgfältig zu bewerten und klare Richtlinien und Prozesse für die Handhabung

von Korrekturen basierend auf den damit verbundenen Risiken und Auswirkungen zu definieren. Indem Sie zwischen kritischen (z.B. Hauptbücher, medizinische Aufzeichnungen) und nicht-kritischen Daten (z.B. Postadressen, Ressourcenwarnungen) unterscheiden, können Sie eine Balance zwischen der Nutzung der Vorteile selbstheilender Datentechniken, wo angemessen, und der Aufrechterhaltung strenger Kontrolle und Überwachung, wo notwendig, finden.

Letztendlich sollte die Entscheidung, selbstheilende Datentechniken auf kritische Daten anzuwenden, in Absprache mit Domänenexperten, Rechtsberatern und anderen relevanten Interessengruppen getroffen werden. Es ist wichtig, die spezifischen Anforderungen, Vorschriften und Risiken im Zusammenhang mit den Daten Ihrer Anwendung zu berücksichtigen und die Datenkorrekturstrategien entsprechend anzupassen.

Fehlerschweregrad

Bei der Anwendung selbstheilender Datentechniken ist es wichtig, den Schweregrad und die Auswirkungen der Datenfehler zu bewerten. Nicht alle Fehler sind gleich, und die angemessene Vorgehensweise kann je nach Schweregrad des Problems variieren.

Kleinere Inkonsistenzen oder Formatierungsprobleme können für eine automatische Korrektur geeignet sein. Beispielsweise kann ein selbstheilender Datenworker, der defektes JSON reparieren soll, fehlende Kommas oder nicht maskierte doppelte Anführungszeichen behandeln, ohne die Bedeutung oder Struktur der Daten wesentlich zu verändern. Diese Art von Fehlern lässt sich oft unkompliziert korrigieren und hat minimale Auswirkungen auf die allgemeine Datenintegrität.

Schwerwiegendere Fehler jedoch, die die Bedeutung oder Integrität der Daten grundlegend verändern, erfordern möglicherweise einen anderen Ansatz. In solchen Fällen reichen automatisierte Korrekturen eventuell nicht aus, und menschliches Eingreifen kann erforderlich sein, um die Genauigkeit und Gültigkeit der Daten sicherzustellen.

Hier kommt das Konzept zum Tragen, KI selbst zur Bestimmung der Fehlerschwere einzusetzen. Durch die Nutzung der Fähigkeiten von KI-Modellen können wir selbstheilende Datenarbeiter entwickeln, die nicht nur Fehler korrigieren, sondern auch deren Schweregrad bewerten und fundierte Entscheidungen über den Umgang mit ihnen treffen.

Betrachten wir beispielsweise einen selbstheilenden Datenarbeiter, der für die Korrektur von Inkonsistenzen in Daten zuständig ist, die in eine Kundendatenbank einfließen. Der Arbeiter kann so konzipiert werden, dass er die Daten analysiert und potenzielle Fehler identifiziert, wie etwa fehlende oder widersprüchliche Informationen. Anstatt jedoch alle Fehler automatisch zu korrigieren, kann der Arbeiter mit zusätzlichen Werkzeugaufrufen ausgestattet werden, die es ihm ermöglichen, schwerwiegende Fehler zur manuellen Überprüfung zu markieren.

Hier ist ein Beispiel, wie dies implementiert werden kann:

```
1 class CustomerDataReviewer
2   include Raix::ChatCompletion
3   include Raix::FunctionDeclarations
4
5   attr_accessor :customer
6
7   function :flag_for_review, reason: { type: "string" } do |params|
8     AdminNotifier.review_request(customer, params[:reason])
9   end
10
11  def initialize(customer)
12    self.customer = customer
13  end
14
15  def call(customer_data)
16    transcript << {
17      system: "You are a customer data reviewer. Your task is to identify
18        and correct inconsistencies in customer data.
19
20        < additional instructions here... >
21
22        If you encounter severe errors that require human review, use the
```

```

23     `flag_for_review` tool to flag the data for manual intervention." }
24
25     transcript << { user: customer.to_json }
26     transcript << { assistant: "Reviewed/corrected data:\n```\n" }
27
28     self.stop = ["```"]
29
30     chat_completion(json: true).then do |result|
31         return if result.blank?
32
33         customer.update(result)
34     end
35 end
36 end

```

In diesem Beispiel ist der `CustomerDataHealer-Worker` darauf ausgelegt, Inkonsistenzen in Kundendaten zu erkennen und zu korrigieren. Auch hier verwenden wir [Response Fencing](#) und [Ventriloquist](#), um strukturierte Ausgaben zu erhalten. Wichtig ist, dass die Systemanweisung des Workers Instruktionen enthält, die Funktion `flag_for_review` zu verwenden, wenn schwerwiegende Fehler auftreten.

Wenn der Worker die Kundendaten verarbeitet, analysiert er die Daten und versucht, etwaige Inkonsistenzen zu korrigieren. Wenn der Worker feststellt, dass die Fehler schwerwiegend sind und menschliches Eingreifen erfordern, kann er das `flag_for_review`-Tool verwenden, um die Daten zu kennzeichnen und einen Grund für die Kennzeichnung anzugeben.

Die `chat_completion`-Methode wird mit `json: true` aufgerufen, um die korrigierten Kundendaten als JSON zu parsen. Es gibt keine Möglichkeit für eine Schleife nach einem Funktionsaufruf, daher wird das Ergebnis leer sein, wenn `flag_for_review` aufgerufen wurde. Andernfalls wird der Kunde mit den überprüften und möglicherweise korrigierten Daten aktualisiert.

Durch die Integration der Fehlerschwerebewertung und die Option, Daten für die manuelle Überprüfung zu kennzeichnen, wird der selbstheilende Daten-Worker intelligenter und anpassungsfähiger. Er kann kleinere Fehler automatisch behandeln

und gleichzeitig schwerwiegende Fehler an menschliche Experten zur manuellen Intervention eskalieren.

Die spezifischen Kriterien zur Bestimmung der Fehlerschwere können in der Anweisung des Workers basierend auf dem Domänenwissen und den geschäftlichen Anforderungen definiert werden. Faktoren wie die Auswirkungen auf die Datenintegrität, das Potenzial für Datenverlust oder -beschädigung und die Folgen falscher Daten können bei der Bewertung der Schwere berücksichtigt werden.

Durch den Einsatz von KI zur Bewertung der Fehlerschwere und die Bereitstellung von Optionen für menschliches Eingreifen können selbstheilende Datentechniken eine Balance zwischen Automatisierung und Aufrechterhaltung der Datengenauigkeit schaffen. Dieser Ansatz stellt sicher, dass kleinere Fehler effizient korrigiert werden, während schwerwiegende Fehler die notwendige Aufmerksamkeit und Expertise von menschlichen Prüfern erhalten.

Domänenkomplexität

Bei der Betrachtung der Anwendung von selbstheilenden Datentechniken ist es wichtig, die Komplexität der Datendomäne und die Regeln, die ihre Struktur und Beziehungen bestimmen, zu bewerten. Die Komplexität der Domäne kann die Effektivität und Durchführbarkeit automatisierter Datenkorrekturansätze erheblich beeinflussen.

Selbstheilende Datentechniken funktionieren gut, wenn die Daten klar definierten Mustern und Einschränkungen folgen. In Domänen, in denen die Datenstruktur relativ einfach ist und die Beziehungen zwischen Datenelementen unkompliziert sind, können automatisierte Korrekturen mit hoher Zuverlässigkeit angewendet werden. Zum Beispiel können Formatierungsprobleme oder die Durchsetzung grundlegender Datentypbeschränkungen oft effektiv von selbstheilenden Daten-Workern behandelt werden.

Mit zunehmender Komplexität der Datendomäne wachsen jedoch auch die Herausforderungen bei der automatisierten Datenkorrektur. In Domänen mit

komplexer Geschäftslogik, komplexen Beziehungen zwischen Datenentitäten oder domänenspezifischen Regeln und Ausnahmen können selbstheilende Datentechniken möglicherweise nicht alle Nuancen erfassen und unbeabsichtigte Konsequenzen verursachen.

Betrachten wir ein Beispiel einer komplexen Domäne: ein Finanzhandelssystem. In dieser Domäne umfassen die Daten verschiedene Finanzinstrumente, Marktdaten, Handelsregeln und regulatorische Anforderungen. Die Beziehungen zwischen verschiedenen Datenelementen können komplex sein, und die Regeln für Datenvalidität und Konsistenz können hochspezifisch für die Domäne sein.

In einer solch komplexen Domäne müsste ein selbstheilender Daten-Worker, der mit der Korrektur von Inkonsistenzen in Handelsdaten beauftragt ist, ein tiefes Verständnis der domänenspezifischen Regeln und Einschränkungen haben. Er müsste Faktoren wie Marktvorschriften, Handelslimits, Risikoberechnungen und Abwicklungsverfahren berücksichtigen. Automatisierte Korrekturen können in diesem Kontext möglicherweise nicht die volle Komplexität der Domäne erfassen und könnten versehentlich Fehler einführen oder domänenspezifische Regeln verletzen.

Um die Herausforderungen der Domänenkomplexität zu bewältigen, können selbstheilende Datentechniken durch die Integration von domänenspezifischem Wissen und Regeln in die KI-Modelle und Worker verbessert werden. Dies kann durch folgende Techniken erreicht werden:

1. **Domänenspezifisches Training:** Die für selbstheilende Daten verwendeten KI-Modelle können auf domänenspezifischen Datensätzen trainiert oder sogar feinabgestimmt werden, die die Feinheiten und Regeln der jeweiligen Domäne erfassen. Indem die Modelle repräsentativen Daten und Szenarien ausgesetzt werden, können sie die domänenspezifischen Muster, Einschränkungen und Ausnahmen lernen.
2. **Regelbasierte Einschränkungen:** Selbstheilende Daten-Worker können mit expliziten regelbasierten Einschränkungen erweitert werden, die

domänenspezifisches Wissen kodieren. Diese Regeln können von Domänenexperten definiert und in den Datenkorrekturprozess integriert werden. Die KI-Modelle können diese Regeln dann nutzen, um ihre Entscheidungen zu steuern und die Einhaltung domänenspezifischer Anforderungen sicherzustellen.

3. **Zusammenarbeit mit Domänenexperten:** In komplexen Domänen ist es entscheidend, Domänenexperten in die Gestaltung und Entwicklung von selbstheilenden Datentechniken einzubeziehen. Domänenexperten können wertvolle Einblicke in die Feinheiten der Daten, die Geschäftsregeln und die möglichen Randfälle liefern. Ihr Wissen kann in die KI-Modelle und Worker integriert werden, um die Genauigkeit und Zuverlässigkeit automatisierter Datenkorrekturen unter Verwendung von [Human In The Loop](#)-Mustern zu verbessern.
4. **Inkrementeller und iterativer Ansatz:** Bei komplexen Domänen ist es oft vorteilhaft, einen inkrementellen und iterativen Ansatz für selbstheilende Daten zu wählen. Anstatt zu versuchen, Korrekturen für die gesamte Domäne auf einmal zu automatisieren, konzentriert man sich auf spezifische Teildomänen oder Datenkategorien, bei denen die Regeln und Einschränkungen gut verstanden sind. Der Umfang der selbstheilenden Techniken wird schrittweise erweitert, während das Verständnis der Domäne wächst und sich die Techniken als effektiv erweisen.

Durch die Berücksichtigung der Komplexität der Datendomäne und die Integration von domänenspezifischem Wissen in selbstheilende Datentechniken können Sie ein Gleichgewicht zwischen Automatisierung und Genauigkeit erreichen. Es ist wichtig zu erkennen, dass selbstheilende Daten keine Universallösung darstellen und dass der Ansatz auf die spezifischen Anforderungen und Herausforderungen jeder Domäne zugeschnitten werden sollte.

In komplexen Domänen kann ein hybrider Ansatz, der selbstheilende Datentechniken mit menschlicher Expertise und Aufsicht kombiniert, am effektivsten sein.

Automatisierte Korrekturen können Routine- und klar definierte Fälle behandeln, während komplexe Szenarien oder Ausnahmen für menschliche Überprüfung und Intervention markiert werden können. Dieser kollaborative Ansatz stellt sicher, dass die Vorteile der Automatisierung realisiert werden, während die notwendige Kontrolle und Genauigkeit in komplexen Datendomänen gewährleistet bleibt.

Erklärbarkeit und Transparenz

Erklärbarkeit bezieht sich auf die Fähigkeit, die Argumentation hinter den Entscheidungen von KI-Modellen zu verstehen und zu interpretieren, während Transparenz die klare Sichtbarkeit des Datenkorrekturprozesses beinhaltet.

In vielen Kontexten müssen Datenänderungen nachvollziehbar und begründbar sein. Interessenvertreter, einschließlich Geschäftsanwender, Prüfer und Regulierungsbehörden, können Erklärungen dafür verlangen, warum bestimmte Datenkorrekturen vorgenommen wurden und wie die KI-Modelle zu diesen Entscheidungen gekommen sind. Dies ist besonders wichtig in Bereichen, in denen Datengenauigkeit und -integrität bedeutende Auswirkungen haben, wie Finanzen, Gesundheitswesen und rechtliche Angelegenheiten.

Um dem Bedarf an Erklärbarkeit und Transparenz gerecht zu werden, sollten selbstheilende Datentechniken Mechanismen enthalten, die Einblicke in den Entscheidungsprozess von KI-Modellen gewähren. Dies kann durch verschiedene Ansätze erreicht werden:

1. **Gedankenkette:** Indem das Modell aufgefordert wird, sein Denken “laut” zu erklären, bevor es Änderungen an Daten vornimmt, kann das Verständnis des Entscheidungsprozesses erleichtert werden und es können menschenlesbare Erklärungen für die vorgenommenen Korrekturen generiert werden. Der Kompromiss ist eine etwas höhere Komplexität bei der Trennung der Erklärung von der strukturierten Datenausgabe, was durch... angegangen werden kann.

2. **Erklärungsgenerierung:** Selbstheilende Datenarbeiter können mit der Fähigkeit ausgestattet werden, menschenlesbare Erklärungen für ihre Korrekturen zu generieren. Dies kann erreicht werden, indem das Modell seinen Entscheidungsprozess als leicht verständliche Erklärungen *direkt in die Daten integriert* ausgibt. Zum Beispiel könnte ein selbstheilender Datenarbeiter einen Bericht erstellen, der die spezifischen Dateninkonsistenzen aufzeigt, die er identifiziert hat, die angewandten Korrekturen und die Begründung für diese Korrekturen.
3. **Merkmalsbedeutung:** KI-Modelle können mit Informationen über die Bedeutung verschiedener Merkmale oder Attribute im Datenkorrekturprozess als Teil ihrer Direktiven ausgestattet werden. Diese Direktiven können wiederum den menschlichen Interessenvertretern zugänglich gemacht werden. Durch die Identifizierung der Schlüsselfaktoren, die die Entscheidungen des Modells beeinflussen, können Interessenvertreter Einblicke in die Begründung der Korrekturen gewinnen und deren Gültigkeit bewerten.
4. **Protokollierung und Prüfung:** Die Implementierung umfassender Protokollierungs- und Prüfmechanismen ist entscheidend für die Aufrechterhaltung der Transparenz im selbstheilenden Datenprozess. Jede von KI-Modellen vorgenommene Datenkorrektur sollte protokolliert werden, einschließlich der ursprünglichen Daten, der korrigierten Daten und der spezifischen durchgeführten Aktionen. Dieser Prüfpfad ermöglicht eine retrospektive Analyse und bietet eine klare Aufzeichnung der an den Daten vorgenommenen Änderungen.
5. **Mensch-in-der-Schleife-Ansatz:** Die Einbindung eines Mensch-in-der-Schleife-Ansatzes kann die Erklärbarkeit und Transparenz von selbstheilenden Datentechniken verbessern. Durch die Einbeziehung menschlicher Experten in die Überprüfung und Validierung von KI-generierten Korrekturen können Organisationen sicherstellen, dass die Korrekturen mit dem Domänenwissen und den geschäftlichen Anforderungen übereinstimmen. Die menschliche Aufsicht

fügt eine zusätzliche Ebene der Verantwortlichkeit hinzu und ermöglicht die Identifizierung potenzieller Verzerrungen oder Fehler in den KI-Modellen.

6. **Kontinuierliche Überwachung und Bewertung:** Die regelmäßige Überwachung und Bewertung der Leistung von selbstheilenden Datentechniken ist wesentlich für die Aufrechterhaltung von Transparenz und Vertrauen. Durch die Beurteilung der Genauigkeit und Effektivität der KI-Modelle im Laufe der Zeit können Organisationen Abweichungen oder Anomalien identifizieren und Korrekturmaßnahmen ergreifen. Kontinuierliche Überwachung hilft sicherzustellen, dass der selbstheilende Datenprozess zuverlässig bleibt und mit den gewünschten Ergebnissen übereinstimmt.

Erklärbarkeit und Transparenz sind kritische Überlegungen bei der Implementierung von selbstheilenden Datentechniken. Durch die Bereitstellung klarer Erklärungen für Datenkorrekturen, die Aufrechterhaltung umfassender Prüfpfade und die Einbeziehung menschlicher Aufsicht können Organisationen Vertrauen in den selbstheilenden Datenprozess aufbauen und sicherstellen, dass die an den Daten vorgenommenen Änderungen gerechtfertigt und mit den Geschäftszielen abgestimmt sind.

Es ist wichtig, ein Gleichgewicht zwischen den Vorteilen der Automatisierung und dem Bedarf an Transparenz zu finden. Während selbstheilende Datentechniken die Datenqualität und Effizienz erheblich verbessern können, sollte dies nicht auf Kosten der Sichtbarkeit und Kontrolle über den Datenkorrekturprozess gehen. Durch die Gestaltung selbstheilender Datenarbeiter mit Fokus auf Erklärbarkeit und Transparenz können Organisationen die Kraft der KI nutzen und gleichzeitig das notwendige Maß an Verantwortlichkeit und Vertrauen in die Daten aufrechterhalten.

Unbeabsichtigte Folgen

Während selbstheilende Datentechniken darauf abzielen, die Datenqualität und -konsistenz zu verbessern, ist es wichtig, sich der möglichen unbeabsichtigten Folgen bewusst zu sein. Automatisierte Korrekturen können, wenn sie nicht sorgfältig

konzipiert und überwacht werden, unbeabsichtigt die Bedeutung oder den Kontext der Daten verändern und zu nachgelagerten Problemen führen.

Eines der Hauptrisiken selbstheilender Daten ist die Einführung von Verzerrungen oder Fehlern im Datenkorrekturprozess. KI-Modelle können, wie jedes andere Softwaresystem auch, Verzerrungen unterliegen, die in den Trainingsdaten vorhanden sind oder durch die Gestaltung der Algorithmen eingeführt werden. Wenn diese Verzerrungen nicht identifiziert und gemildert werden, können sie sich durch den selbstheilenden Datenprozess fortpflanzen und zu verzerrten oder falschen Datenänderungen führen.

Betrachten wir zum Beispiel einen selbstheilenden Datenarbeiter, der mit der Korrektur von Inkonsistenzen in demografischen Kundendaten beauftragt ist. Wenn das KI-Modell Voreingenommenheiten aus historischen Daten gelernt hat, wie etwa die Verknüpfung bestimmter Berufe oder Einkommensniveaus mit spezifischen Geschlechtern oder ethnischen Gruppen, könnte es falsche Annahmen treffen und die Daten so modifizieren, dass diese Vorurteile verstärkt werden. Dies kann zu ungenauen Kundenprofilen, fehlgeleiteten Geschäftsentscheidungen und möglicherweise diskriminierenden Ergebnissen führen.

Eine weitere potenzielle unbeabsichtigte Folge ist der Verlust wertvoller Informationen oder Kontexte während des Datenkorrekturprozesses. Selbstheilende Datentechniken konzentrieren sich oft darauf, Daten zu standardisieren und zu normalisieren, um Konsistenz zu gewährleisten. In manchen Fällen können die ursprünglichen Daten jedoch Nuancen, Ausnahmen oder kontextuelle Informationen enthalten, die für das Verständnis des Gesamtbildes wichtig sind. Automatisierte Korrekturen, die blind eine Standardisierung erzwingen, können diese wertvollen Informationen unbeabsichtigt entfernen oder verschleiern.

Stellen Sie sich zum Beispiel einen selbstheilenden Datenarbeiter vor, der für die Korrektur von Inkonsistenzen in medizinischen Aufzeichnungen zuständig ist. Wenn der Arbeiter auf die Krankengeschichte eines Patienten mit einer seltenen Erkrankung

oder einem ungewöhnlichen Behandlungsplan stößt, könnte er versuchen, die Daten an ein häufigeres Muster anzupassen. Dabei könnten jedoch die spezifischen Details und der Kontext verloren gehen, die für die genaue Darstellung der einzigartigen Situation des Patienten entscheidend sind. Dieser Informationsverlust kann schwerwiegende Auswirkungen auf die Patientenversorgung und medizinische Entscheidungsfindung haben.

Um die Risiken unbeabsichtigter Folgen zu minimieren, ist es wichtig, bei der Konzeption und Implementierung selbstheilender Datentechniken einen proaktiven Ansatz zu verfolgen:

1. **Gründliche Tests und Validierung:** Vor dem Einsatz selbstheilender Datenarbeiter in der Produktion ist es entscheidend, ihr Verhalten gründlich anhand verschiedener Szenarien zu testen und zu validieren. Dies umfasst Tests mit repräsentativen Datensätzen, die verschiedene Grenzfälle, Ausnahmen und potenzielle Voreingenommenheiten abdecken. Rigorose Tests helfen dabei, unbeabsichtigte Folgen zu erkennen und zu beheben, bevor sie sich auf reale Daten auswirken.
2. **Kontinuierliche Überwachung und Auswertung:** Die Implementierung kontinuierlicher Überwachungs- und Auswertungsmechanismen ist wichtig, um unbeabsichtigte Folgen im Laufe der Zeit zu erkennen und zu minimieren. Die regelmäßige Überprüfung der Ergebnisse selbstheilender Datenprozesse, die Analyse der Auswirkungen auf nachgelagerte Systeme und Entscheidungsfindungen sowie das Einholen von Feedback von Stakeholdern können helfen, negative Auswirkungen zu identifizieren und zeitnahe Korrekturmaßnahmen einzuleiten. Wenn Ihre Organisation über Betriebsdashboards verfügt, ist es wahrscheinlich eine gute Idee, gut sichtbare Metriken zu automatisierten Datenänderungen hinzuzufügen. Noch besser ist es vermutlich, Alarme einzurichten, die bei großen Abweichungen von der normalen Datenänderungsaktivität ausgelöst werden!

3. **Menschliche Aufsicht und Intervention:** Die Aufrechterhaltung menschlicher Aufsicht und die Möglichkeit zur Intervention im selbstheilenden Datenprozess ist entscheidend. Während Automatisierung die Effizienz stark verbessern kann, ist es wichtig, dass menschliche Experten die von KI-Modellen vorgenommenen Korrekturen überprüfen und validieren, besonders in kritischen oder sensiblen Bereichen. Menschliches Urteilsvermögen und Fachwissen können helfen, auftretende unbeabsichtigte Folgen zu erkennen und zu beheben.
4. **Erklärbare KI (XAI) und Transparenz:** Wie im vorherigen Abschnitt diskutiert, kann die Einbindung erklärbarer KI-Techniken und die Gewährleistung von Transparenz im selbstheilenden Datenprozess dazu beitragen, unbeabsichtigte Folgen zu minimieren. Durch die Bereitstellung klarer Erklärungen für Datenkorrekturen und die Führung umfassender Prüfpfade können Organisationen die Argumentation hinter den von KI-Modellen vorgenommenen Änderungen besser verstehen und nachverfolgen.
5. **Inkrementeller und iterativer Ansatz:** Die Anwendung eines inkrementellen und iterativen Ansatzes für selbstheilende Daten kann dazu beitragen, das Risiko unbeabsichtigter Folgen zu minimieren. Anstatt automatisierte Korrekturen sofort auf den gesamten Datensatz anzuwenden, beginnen Sie mit einer Teilmenge der Daten und erweitern Sie den Umfang schrittweise, wenn sich die Techniken als effektiv und zuverlässig erweisen. Dies ermöglicht eine sorgfältige Überwachung und Anpassung während des Prozesses und reduziert die Auswirkungen unbeabsichtigter Folgen.
6. **Zusammenarbeit und Feedback:** Die Einbindung von Stakeholdern aus verschiedenen Bereichen und die Förderung von Zusammenarbeit und Feedback während des selbstheilenden Datenprozesses kann helfen, unbeabsichtigte Folgen zu erkennen und zu beheben. Das regelmäßige Einholen von Input von Domänenexperten, Datenkonsumenten und Endnutzern kann wertvolle Einblicke in die realen Auswirkungen der Datenkorrekturen liefern und mögliche übersehene Probleme aufzeigen.

Durch die proaktive Bewältigung des Risikos unbeabsichtigter Folgen und die Implementierung geeigneter Schutzmaßnahmen können Organisationen die Vorteile selbstheilender Datentechniken nutzen und gleichzeitig potenzielle negative Auswirkungen minimieren. Es ist wichtig, selbstheilende Daten als einen iterativen und kollaborativen Prozess zu betrachten, der kontinuierlich überwacht, ausgewertet und verfeinert wird, um sicherzustellen, dass sie mit den gewünschten Ergebnissen übereinstimmen und die Integrität und Zuverlässigkeit der Daten gewährleisten.

Bei der Erwägung des Einsatzes selbstheilender Datenmuster ist es wichtig, diese Faktoren sorgfältig zu evaluieren und die Vorteile gegen die potenziellen Risiken und Einschränkungen abzuwägen. In manchen Fällen könnte ein hybrider Ansatz, der automatisierte Korrekturen mit menschlicher Aufsicht und Intervention kombiniert, die am besten geeignete Lösung sein.

Es ist auch erwähnenswert, dass selbstheilende Datentechniken nicht als Ersatz für robuste Datenvalidierung, Eingabebereinigung und Fehlerbehandlungsmechanismen gesehen werden sollten. Diese grundlegenden Praktiken bleiben für die Gewährleistung der Datenintegrität und -sicherheit von entscheidender Bedeutung. Selbstheilende Daten sollten als ergänzender Ansatz betrachtet werden, der diese bestehenden Maßnahmen erweitern und verbessern kann.

Letztendlich hängt die Entscheidung für den Einsatz selbstheilender Datenmuster von den spezifischen Anforderungen, Einschränkungen und Prioritäten Ihrer Anwendung ab. Durch sorgfältige Berücksichtigung der oben genannten Überlegungen und deren Abstimmung mit den Zielen und der Architektur Ihrer Anwendung können Sie fundierte Entscheidungen darüber treffen, wann und wie selbstheilende Datentechniken effektiv eingesetzt werden können.

Kontextbezogene Inhaltsgenerierung



Muster der kontextbezogenen Inhaltsgenerierung nutzen die Leistungsfähigkeit von Large Language Models (LLMs), um dynamische und kontextspezifische Inhalte innerhalb von Anwendungen zu generieren. Diese Kategorie von Mustern erkennt die Bedeutung der Bereitstellung personalisierter und relevanter Inhalte für Benutzer, basierend auf deren spezifischen Bedürfnissen, Präferenzen und sogar früheren sowie aktuellen Interaktionen mit der Anwendung.

In diesem Ansatz bezieht sich “Inhalt” sowohl auf primäre Inhalte (z.B. Blogbeiträge, Artikel usw.) als auch auf Meta-Inhalte, wie etwa Empfehlungen zu primären

Inhalten.

Muster der kontextbezogenen Inhaltsgenerierung können eine entscheidende Rolle bei der Verbesserung Ihrer Benutzerinteraktionsniveaus spielen, maßgeschneiderte Erfahrungen bieten und Aufgaben der Inhaltserstellung sowohl für Sie als auch für Ihre Benutzer automatisieren. Durch die Nutzung der in diesem Kapitel beschriebenen Muster können Sie Anwendungen erstellen, die Inhalte dynamisch generieren und sich in Echtzeit an den Kontext und die Eingaben anpassen.

Die Muster funktionieren durch die Integration von LLMs in die Ausgaben der Anwendung, von der Benutzeroberfläche (manchmal als “Chrome” bezeichnet) über E-Mails und andere Benachrichtigungsformen bis hin zu allen Arten von Inhaltsgenerierungspipelines.

Wenn ein Benutzer mit der Anwendung interagiert oder eine bestimmte Inhaltsanfrage auslöst, erfasst die Anwendung den relevanten Kontext, wie etwa Benutzerpräferenzen, vorherige Interaktionen oder spezifische Eingabeaufforderungen. Diese kontextuellen Informationen werden dann zusammen mit allen erforderlichen Vorlagen oder Richtlinien in das LLM eingespeist und zur Erzeugung von Textausgaben verwendet, die sonst entweder fest codiert, in einer Datenbank gespeichert oder algorithmisch generiert werden müssten.

Die von LLMs generierten Inhalte können verschiedene Formen annehmen, wie personalisierte Empfehlungen, dynamische Produktbeschreibungen, maßgeschneiderte E-Mail-Antworten oder sogar vollständige Artikel oder Blogbeiträge. Eine der radikalsten Anwendungen dieser Inhalte, die ich vor über einem Jahr entwickelt habe, ist die dynamische Generierung von UI-Elementen wie Formularbezeichnungen, Tooltips und anderen Arten von erläuterndem Text.

Personalisierung

Einer der wichtigsten Vorteile von Mustern der kontextbezogenen Inhaltsgenerierung ist die Fähigkeit, hochgradig personalisierte Erfahrungen für Benutzer bereitzustellen. Durch die Generierung von Inhalten basierend auf benutzerspezifischem Kontext ermöglichen diese Muster Anwendungen, Inhalte auf die individuellen Interessen, Präferenzen und Interaktionen der Benutzer zuzuschneiden.

Personalisierung geht über das einfache Einfügen eines Benutzernamens in generische Inhalte hinaus. Es beinhaltet die Nutzung des reichhaltigen Kontexts, der über jeden Benutzer verfügbar ist, um Inhalte zu generieren, die mit ihren spezifischen Bedürfnissen und Wünschen resonieren. Dieser Kontext kann eine Vielzahl von Faktoren umfassen, wie:

1. **Benutzerprofilinformationen:** Auf der allgemeinsten Ebene der Anwendung dieser Technik können demografische Daten, Interessen, Präferenzen und andere Profilattribute genutzt werden, um Inhalte zu generieren, die mit dem Hintergrund und den Eigenschaften des Benutzers übereinstimmen.
2. **Verhaltensdaten:** Die vergangenen Interaktionen eines Benutzers mit der Anwendung, wie angesehene Seiten, angeklickte Links oder gekaufte Produkte, können wertvolle Einblicke in ihr Verhalten und ihre Interessen liefern. Diese Daten können verwendet werden, um Inhaltsvorschläge zu generieren, die ihre Interaktionsmuster widerspiegeln und ihre zukünftigen Bedürfnisse vorhersagen.
3. **Kontextuelle Faktoren:** Der aktuelle Kontext des Benutzers, wie Standort, Gerät, Tageszeit oder sogar das Wetter, kann den Inhaltsgenerierungsprozess beeinflussen. Beispielsweise könnte eine Reise-Anwendung über einen KI-Worker verfügen, der personalisierte Empfehlungen basierend auf dem aktuellen Standort des Benutzers und den vorherrschenden Wetterbedingungen generieren kann.

Durch die Nutzung dieser kontextuellen Faktoren ermöglichen Muster der kontextbezogenen Inhaltsgenerierung Anwendungen, Inhalte bereitzustellen, die sich für jeden einzelnen Benutzer maßgeschneidert anfühlen. Diese Ebene der Personalisierung hat mehrere bedeutende Vorteile:

1. **Erhöhte Interaktion:** Personalisierte Inhalte fesseln die Aufmerksamkeit der Benutzer und halten sie bei der Anwendung engagiert. Wenn Benutzer das Gefühl haben, dass die Inhalte relevant sind und direkt ihre Bedürfnisse ansprechen, verbringen sie mit höherer Wahrscheinlichkeit mehr Zeit mit der Interaktion mit der Anwendung und der Erkundung ihrer Funktionen.
2. **Verbesserte Benutzerzufriedenheit:** Personalisierte Inhalte zeigen, dass die Anwendung die einzigartigen Anforderungen des Benutzers versteht und berücksichtigt. Durch die Bereitstellung von Inhalten, die hilfreich, informativ und auf ihre Interessen abgestimmt sind, kann die Anwendung die Benutzerzufriedenheit steigern und eine stärkere Verbindung zu ihren Benutzern aufbauen.
3. **Höhere Konversionsraten:** Im Kontext von E-Commerce- oder Marketing-Anwendungen können personalisierte Inhalte die Konversionsraten erheblich beeinflussen. Indem Benutzern Produkte, Angebote oder Empfehlungen präsentiert werden, die auf ihre Präferenzen und ihr Verhalten zugeschnitten sind, kann die Anwendung die Wahrscheinlichkeit erhöhen, dass Benutzer gewünschte Aktionen ausführen, wie einen Kauf zu tätigen oder sich für einen Dienst anzumelden.

Produktivität

Muster der kontextbezogenen Inhaltsgenerierung können bestimmte Arten der Produktivität erheblich steigern, indem sie den Bedarf an manueller Inhaltsgenerierung und -bearbeitung in kreativen Prozessen reduzieren. Durch die Nutzung der

Leistungsfähigkeit von LLMs können Sie qualitativ hochwertige Inhalte im großen Maßstab generieren und dabei Zeit und Aufwand sparen, die Ihre Content-Ersteller und Entwickler sonst für mühsame manuelle Arbeit aufwenden müssten.

Traditionell müssen Content-Ersteller Inhalte recherchieren, schreiben, bearbeiten und formatieren, um sicherzustellen, dass diese den Anforderungen der Anwendung und den Erwartungen der Nutzer entsprechen. Dieser Prozess kann zeitaufwendig und ressourcenintensiv sein, besonders wenn die Menge der Inhalte wächst.

Mit Mustern der kontextuellen Inhaltsgenerierung kann der Content-Erstellungsprozess jedoch weitgehend automatisiert werden. LLMs können auf Basis der bereitgestellten Prompts und Richtlinien kohärente, grammatikalisch korrekte und kontextuell relevante Inhalte generieren. Diese Automatisierung bietet mehrere Produktivitätsvorteile:

1. **Reduzierter manueller Aufwand:** Durch die Delegation von Inhaltsgenerierungsaufgaben an LLMs können sich Content-Ersteller auf übergeordnete Aufgaben wie Content-Strategie, Ideenfindung und Qualitätssicherung konzentrieren. Sie können dem LLM den notwendigen Kontext, Vorlagen und Richtlinien zur Verfügung stellen und ihm die eigentliche Inhaltsgenerierung überlassen. Dies reduziert den manuellen Aufwand für das Schreiben und Bearbeiten und ermöglicht es Content-Erstellern, produktiver und effizienter zu arbeiten.
2. **Schnellere Inhaltserstellung:** LLMs können Inhalte viel schneller generieren als menschliche Autoren. Mit den richtigen Prompts und Richtlinien kann ein LLM mehrere Inhaltsstücke in wenigen Sekunden oder Minuten produzieren. Diese Geschwindigkeit ermöglicht es Anwendungen, Inhalte in einem viel schnelleren Tempo zu generieren und mit den Anforderungen der Nutzer und der sich ständig verändernden digitalen Landschaft Schritt zu halten.

Führt schnellere Inhaltserstellung zu einer “Tragik der Allmende” Situation, in der das Internet in Inhalten ertrinkt, die niemand liest? Leider vermute ich, dass die Antwort ja lautet.

3. **Konsistenz und Qualität:** LLMs können Inhalte problemlos so überarbeiten, dass sie in Stil, Ton und Qualität konsistent sind. Mit klaren Richtlinien und Beispielen können bestimmte Arten von Anwendungen (z.B. Nachrichtenredaktionen, PR etc.) sicherstellen, dass ihre von Menschen erstellten Inhalte mit ihrer Markenstimme übereinstimmen und den gewünschten Qualitätsstandards entsprechen. Diese Konsistenz reduziert den Bedarf an umfangreichen Überarbeitungen und Korrekturen und spart Zeit und Aufwand im Content-Erstellungsprozess.
4. **Iteration und Optimierung:** Muster der kontextuellen Inhaltsgenerierung ermöglichen schnelle Iteration und Optimierung von Inhalten. Durch Anpassung der Prompts, Vorlagen oder Richtlinien für das LLM können Ihre Anwendungen schnell Inhaltsvariationen generieren und verschiedene Ansätze auf eine automatisierte Weise testen, die in der Vergangenheit nie möglich war. Dieser iterative Prozess ermöglicht schnelleres Experimentieren und Verfeinern von Content-Strategien, was im Laufe der Zeit zu effektiveren und ansprechenderen Inhalten führt. Diese spezielle Technik kann ein echter Game-Changer für Anwendungen wie E-Commerce sein, die von Absprungraten und Engagement leben oder sterben.



Es ist wichtig zu beachten, dass Muster der kontextuellen Inhaltsgenerierung zwar die Produktivität erheblich steigern können, aber den Bedarf an menschlicher Beteiligung nicht vollständig eliminieren. Content-Ersteller und Redakteure spielen weiterhin eine entscheidende Rolle bei der Definition der übergeordneten Content-Strategie, der Anleitung des LLM und der Sicherstellung der Qualität und Angemessenheit der generierten Inhalte.

Durch die Automatisierung der eher repetitiven und zeitaufwendigen Aspekte der Inhaltserstellung setzen Muster der kontextuellen Inhaltsgenerierung wertvolle menschliche Zeit und Ressourcen frei, die für höherwertige Aufgaben eingesetzt werden können. Diese gesteigerte Produktivität ermöglicht es Ihnen, personalisierte und ansprechendere Inhalte für Nutzer bereitzustellen und gleichzeitig die Workflows der Inhaltserstellung zu optimieren.

Schnelle Iteration und Experimentierung

Muster der kontextuellen Inhaltsgenerierung ermöglichen es Ihnen, schnell zu iterieren und mit verschiedenen Inhaltsvariationen zu experimentieren, was eine schnellere Optimierung und Verfeinerung Ihrer Content-Strategie ermöglicht. Sie können in Sekundenschnelle mehrere Versionen von Inhalten generieren, indem Sie einfach den Kontext, die Vorlagen oder die Richtlinien für das Modell anpassen.

Diese Fähigkeit zur schnellen Iteration bietet mehrere wichtige Vorteile:

1. **Testen und Optimierung:** Mit der Möglichkeit, schnell Inhaltsvariationen zu generieren, können Sie einfach verschiedene Ansätze testen und deren Wirksamkeit messen. Sie können zum Beispiel mehrere Versionen einer Produktbeschreibung oder einer Marketing-Botschaft generieren, die jeweils auf ein bestimmtes Nutzersegment oder einen bestimmten Kontext zugeschnitten sind. Durch die Analyse von Nutzerinteraktionsmetriken wie Klickraten oder

Konversionsraten können Sie die effektivsten Inhaltsvariationen identifizieren und Ihre Content-Strategie entsprechend optimieren.

2. **A/B-Tests:** Muster der kontextuellen Inhaltsgenerierung ermöglichen nahtloses A/B-Testing von Inhalten. Sie können zwei oder mehr Variationen von Inhalten generieren und diese zufällig verschiedenen Nutzergruppen präsentieren. Durch den Vergleich der Leistung jeder Variation können Sie feststellen, welche Inhalte bei Ihrer Zielgruppe am besten ankommen. Dieser datengesteuerte Ansatz ermöglicht es Ihnen, fundierte Entscheidungen zu treffen und Ihre Inhalte kontinuierlich zu verfeinern, um die Nutzerinteraktion zu maximieren und Ihre gewünschten Ergebnisse zu erzielen.
3. **Personalisierungsexperimente:** Schnelle Iteration und Experimentierung sind besonders wertvoll, wenn es um Personalisierung geht. Mit Mustern der kontextuellen Inhaltsgenerierung können Sie schnell personalisierte Inhaltsvariationen basierend auf verschiedenen Nutzersegmenten, Präferenzen oder Verhaltensweisen generieren. Durch das Experimentieren mit verschiedenen Personalisierungsstrategien können Sie die effektivsten Ansätze identifizieren, um einzelne Nutzer anzusprechen und maßgeschneiderte Erlebnisse zu bieten.
4. **Anpassung an sich ändernde Trends:** Die Fähigkeit zur schnellen Iteration und zum Experimentieren ermöglicht es Ihnen, agil zu bleiben und sich an verändernde Trends und Benutzerpräferenzen anzupassen. Wenn neue Themen, Schlüsselwörter oder Benutzerverhalten auftauchen, können Sie schnell Inhalte erstellen, die mit diesen Trends übereinstimmen. Durch kontinuierliches Experimentieren und Verfeinern Ihrer Inhalte können Sie relevant bleiben und sich einen Wettbewerbsvorteil in der sich ständig weiterentwickelnden digitalen Landschaft sichern.
5. **Kosteneffektives Experimentieren:** Traditionelles Content-Experimentieren erfordert oft erheblichen Zeit- und Ressourcenaufwand, da Content-Ersteller verschiedene Variationen manuell entwickeln und testen müssen. Mit Kontextbezogenen Inhaltsgenerierungsmustern werden die Kosten

für Experimente jedoch deutlich reduziert. Große Sprachmodelle können Inhaltsvariationen schnell und in großem Umfang generieren, sodass Sie eine breite Palette von Ideen und Ansätzen erkunden können, ohne erhebliche Kosten zu verursachen.

Um das Beste aus schneller Iteration und Experimentieren herauszuholen, ist es wichtig, ein gut definiertes Experimentier-Framework zu haben. Dieses Framework sollte Folgendes umfassen:

- Klare Ziele und Hypothesen für jedes Experiment
- Geeignete Metriken und Tracking-Mechanismen zur Messung der Content-Performance
- Segmentierungs- und Targeting-Strategien, um sicherzustellen, dass relevante Inhaltsvariationen den richtigen Benutzern bereitgestellt werden
- Analyse- und Berichtswerkzeuge zur Ableitung von Erkenntnissen aus den experimentellen Daten
- Ein Prozess zur Integration von Erkenntnissen und Optimierungen in Ihre Content-Strategie

Durch die Einbindung von schneller Iteration und Experimentieren können Sie Ihre Inhalte kontinuierlich verfeinern und optimieren und sicherstellen, dass sie ansprechend, relevant und effektiv bei der Erreichung der Ziele Ihrer Anwendung bleiben. Dieser agile Ansatz zur Inhaltserstellung ermöglicht es Ihnen, der Entwicklung voraus zu sein und außergewöhnliche Benutzererfahrungen zu liefern.

Skalierbarkeit und Effizienz

Mit dem Wachstum von Anwendungen und der steigenden Nachfrage nach personalisierten Inhalten ermöglichen kontextbezogene Inhaltsgenerierungsmuster eine effiziente Skalierung der Inhaltserstellung. Große Sprachmodelle können

Inhalte für eine große Anzahl von Benutzern und Kontexten gleichzeitig generieren, ohne dass die personellen Ressourcen proportional erhöht werden müssen. Diese Skalierbarkeit ermöglicht es Anwendungen, personalisierte Erlebnisse für eine wachsende Benutzerbasis bereitzustellen, ohne ihre Inhaltserstellungskapazitäten zu überlasten.



Beachten Sie, dass kontextbezogene Inhaltsgenerierung effektiv zur Internationalisierung Ihrer Anwendung “im laufenden Betrieb” verwendet werden kann. Tatsächlich ist genau das, was ich mit meinem Instant18n Gem gemacht habe, um Olympia in mehr als einem halben Dutzend Sprachen bereitzustellen, obwohl wir noch nicht einmal ein Jahr alt sind.

KI-gestützte Lokalisierung

Wenn Sie mir einen Moment der Prahlerei erlauben: Ich denke, dass meine Instant18n-Bibliothek für Rails-Anwendungen ein bahnbrechendes Beispiel für das Muster der “Kontextbezogenen Inhaltsgenerierung” in Aktion ist und das transformative Potenzial von KI in der Anwendungsentwicklung zeigt. Dieses Gem nutzt die Leistungsfähigkeit von OpenAIs GPT-Sprachmodell, um die Art und Weise zu revolutionieren, wie Internationalisierung und Lokalisierung in Rails-Anwendungen gehandhabt werden.

Traditionell erfordert die Internationalisierung einer Rails-Anwendung die manuelle Definition von Übersetzungsschlüsseln und die Bereitstellung entsprechender Übersetzungen für jede unterstützte Sprache. Dieser Prozess kann zeitaufwändig, ressourcenintensiv und anfällig für Inkonsistenzen sein. Mit dem Instant18n Gem wird das Paradigma der Lokalisierung jedoch vollständig neu definiert.

Durch die Integration eines großen Sprachmodells ermöglicht das Instant18n Gem die Generierung von Übersetzungen im laufenden Betrieb, basierend auf dem Kontext und der Bedeutung des Textes. Anstatt sich auf vordefinierte Übersetzungsschlüssel und

statische Übersetzungen zu verlassen, übersetzt das Gem Texte dynamisch mithilfe der KI. Dieser Ansatz bietet mehrere wichtige Vorteile:

1. **Nahtlose Lokalisierung:** Mit dem Instant18n Gem müssen Entwickler keine Übersetzungsdateien für jede unterstützte Sprache mehr manuell definieren und pflegen. Das Gem generiert automatisch Übersetzungen basierend auf dem bereitgestellten Text und der gewünschten Zielsprache, was den Lokalisierungsprozess mühelos und nahtlos macht.
2. **Kontextuelle Genauigkeit:** KI kann genügend Kontext erhalten, um die Nuancen des zu übersetzenden Textes zu erfassen. Sie kann den umgebenden Kontext, Redewendungen und kulturelle Referenzen berücksichtigen, um Übersetzungen zu generieren, die präzise, natürlich klingend und kontextuell angemessen sind.
3. **Umfangreiche Sprachunterstützung:** Das Instant18n Gem nutzt die umfassenden Kenntnisse und linguistischen Fähigkeiten von GPT und ermöglicht Übersetzungen in eine große Auswahl an Sprachen. Von gängigen Sprachen wie Spanisch und Französisch bis hin zu selteneren oder fiktiven Sprachen wie Klingonisch und Elbisch kann das Gem eine Vielzahl von Übersetzungsanforderungen bewältigen.
4. **Flexibilität und Kreativität:** Das Gem geht über traditionelle Sprachübersetzungen hinaus und ermöglicht kreative und unkonventionelle Lokalisierungsoptionen. Entwickler können Text in verschiedene Stile, Dialekte oder sogar fiktive Sprachen übersetzen und eröffnen damit neue Möglichkeiten für einzigartige Benutzererlebnisse und ansprechende Inhalte.
5. **Leistungsoptimierung:** Das Instant18n Gem integriert Caching-Mechanismen zur Verbesserung der Leistung und Reduzierung des Overheads bei wiederholten Übersetzungen. Übersetzte Texte werden zwischengespeichert, sodass nachfolgende Anfragen für dieselbe Übersetzung schnell und ohne redundante API-Aufrufe bereitgestellt werden können.

Das Instant18n Gem veranschaulicht die Leistungsfähigkeit des Musters

“Kontextbezogene Inhaltsgenerierung”, indem es KI nutzt, um lokalisierte Inhalte dynamisch zu generieren. Es zeigt, wie KI in die Kernfunktionalität einer Rails-Anwendung integriert werden kann und dabei die Art und Weise verändert, wie Entwickler an Internationalisierung und Lokalisierung herangehen.

Durch die Eliminierung der Notwendigkeit manueller Übersetzungsverwaltung und die Ermöglichung von Echtzeit-Übersetzungen basierend auf dem Kontext spart das Instant18n Gem Entwicklern erheblich Zeit und Aufwand. Es ermöglicht ihnen, sich auf die Entwicklung der Kernfunktionen ihrer Anwendung zu konzentrieren, während sichergestellt wird, dass der Lokalisierungsaspekt nahtlos und präzise gehandhabt wird.

Die Bedeutung von Benutzertests und Feedback

Behalten Sie schließlich immer die Bedeutung von Benutzertests und Feedback im Auge. Es ist entscheidend zu validieren, dass die kontextbezogene Inhaltsgenerierung die Erwartungen der Benutzer erfüllt und mit den Zielen der Anwendung übereinstimmt. Entwickeln und verfeinern Sie generierte Inhalte kontinuierlich basierend auf Benutzereinsichten und Analysen. Wenn Sie dynamische Inhalte in einem Umfang generieren, der eine manuelle Validierung durch Sie und Ihr Team unmöglich macht, sollten Sie Feedback-Mechanismen einbauen, die es Benutzern ermöglichen, seltsame oder falsche Inhalte zu melden, zusammen mit einer Erklärung warum. Dieses wertvolle Feedback kann sogar an einen KI-Worker weitergeleitet werden, der damit beauftragt ist, Anpassungen an der Komponente vorzunehmen, die den Inhalt generiert hat!

Generative UI



Aufmerksamkeit ist heutzutage ein so knappes Gut, dass effektives Nutzerengagement inzwischen Softwareerlebnisse erfordert, die nicht nur nahtlos und intuitiv, sondern auch hochgradig auf individuelle Bedürfnisse, Präferenzen und Kontexte zugeschnitten sind. Infolgedessen stehen Designer und Entwickler zunehmend vor der Herausforderung, Benutzeroberflächen zu erstellen, die sich *im großen Maßstab* an die einzigartigen Anforderungen jedes Nutzers anpassen können.

Generative UI (GenUI) ist ein wahrhaft revolutionärer Ansatz für das Design von Benutzeroberflächen, der die Leistungsfähigkeit von Large Language Models (LLMs) nutzt, um hochgradig personalisierte und dynamische Nutzererlebnisse in Echtzeit zu erstellen. Ich wollte Ihnen in diesem Buch unbedingt zumindest eine Einführung in GenUI geben, da ich glaube, dass es eine der vielversprechendsten neuen Möglichkeiten ist, die derzeit im Bereich des Anwendungsdesigns und der Frameworks existieren. Ich bin überzeugt, dass in dieser speziellen Nische Dutzende oder mehr neue erfolgreiche

kommerzielle und Open-Source-Projekte entstehen werden.

Im Kern kombiniert GenUI die Prinzipien der [Kontextbasierten Inhaltsgenerierung](#) mit fortschrittlichen KI-Techniken, um Benutzeroberflächen-Elemente wie Text, Bilder und Layouts dynamisch zu generieren, basierend auf einem tiefgehenden Verständnis des Nutzerkontexts, der Präferenzen und Ziele. GenUI ermöglicht es Designern und Entwicklern, Schnittstellen zu erstellen, die sich als Reaktion auf Nutzerinteraktionen anpassen und weiterentwickeln und damit ein Maß an Personalisierung bieten, das bisher unerreichbar war.

GenUI stellt eine grundlegende Veränderung in unserem Ansatz zum Design von Benutzeroberflächen dar. Statt für die Masse zu designen, ermöglicht GenUI das Design für den Einzelnen. Personalisierte Inhalte und Schnittstellen haben das Potenzial, Nutzererlebnisse zu schaffen, die bei jedem Nutzer auf einer tieferen Ebene resonieren und dadurch Engagement, Zufriedenheit und Loyalität steigern.

Als Technologie an vorderster Front ist der Übergang zu GenUI voller konzeptioneller und praktischer Herausforderungen. Die Integration von KI in den Designprozess und die Sicherstellung, dass die generierten Schnittstellen nicht nur personalisiert, sondern auch nutzbar, zugänglich und mit dem Gesamtmarkenauftritt und der Nutzererfahrung übereinstimmen - all diese Herausforderungen machen GenUI zu einer Aufgabe für die wenigen, nicht die vielen. Darüber hinaus wirft der Einsatz von KI Fragen zum Datenschutz, zur Transparenz und möglicherweise sogar zu ethischen Implikationen auf.

Trotz der Herausforderungen haben personalisierte Erlebnisse im großen Maßstab die Kraft, die Art und Weise, wie wir mit digitalen Produkten und Dienstleistungen interagieren, vollständig zu transformieren. Es eröffnet Möglichkeiten zur Schaffung inklusiver und zugänglicher Schnittstellen, die den vielfältigen Bedürfnissen der Nutzer gerecht werden, unabhängig von ihren Fähigkeiten, Hintergründen oder Präferenzen.

In diesem Kapitel werden wir das Konzept von GenUI erkunden und einige charakteristische Merkmale, zentrale Vorteile und potenzielle Herausforderungen

untersuchen. Wir beginnen mit der grundlegendsten und zugänglichsten Form von GenUI: der Generierung von Textkopien für ansonsten traditionell gestaltete und implementierte Benutzeroberflächen.

Generierung von Texten für Benutzeroberflächen

Textelemente, die in der Benutzeroberfläche Ihrer Anwendung existieren, wie Formularfelder, Tooltips und erklärende Texte, sind typischerweise fest in den Templates oder UI-Komponenten codiert und bieten allen Nutzern ein konsistentes, aber generisches Erlebnis. Mithilfe von kontextbasierten Inhaltsgenerierungsmustern können Sie diese statischen Elemente in dynamische, kontextbewusste und personalisierte Komponenten umwandeln.

Personalisierte Formulare

Formulare sind ein allgegenwärtiger Bestandteil von Web- und Mobile-Anwendungen und dienen als primäres Mittel zur Erfassung von Nutzereingaben. Traditionelle Formulare bieten jedoch oft ein generisches und unpersönliches Erlebnis mit Standardbeschriftungen und -feldern, die nicht immer mit dem spezifischen Kontext oder den Bedürfnissen des Nutzers übereinstimmen. Nutzer füllen mit höherer Wahrscheinlichkeit Formulare aus, die auf ihre Bedürfnisse und Präferenzen zugeschnitten sind, was zu höheren Konversionsraten und größerer Nutzerzufriedenheit führt.

Es ist jedoch wichtig, ein Gleichgewicht zwischen Personalisierung und Konsistenz zu finden. Während die Anpassung von Formularen an einzelne Nutzer vorteilhaft sein kann, ist es entscheidend, ein gewisses Maß an Vertrautheit und Vorhersehbarkeit zu bewahren. Nutzer sollten Formulare auch mit personalisierten Elementen noch leicht erkennen und navigieren können.

Hier sind einige Ideen für personalisierte Formulare zur Inspiration:

Kontextbezogene Feldvorschläge

GenUI kann die vorherigen Interaktionen, Präferenzen und Daten des Nutzers analysieren, um intelligente Feldvorschläge als Vorhersagen anzubieten. Wenn der Nutzer beispielsweise zuvor seine Lieferadresse eingegeben hat, kann das Formular die relevanten Felder automatisch mit seinen gespeicherten Informationen ausfüllen. Dies spart nicht nur Zeit, sondern zeigt auch, dass die Anwendung die Präferenzen des Nutzers versteht und sich daran erinnert.

Moment mal, ist diese Technik nicht auch ohne KI-Einsatz möglich? Natürlich, aber der Reiz, solche Funktionalität mit KI zu steuern, liegt in zweierlei Hinsicht: 1) wie einfach sie sich implementieren lässt und 2) wie widerstandsfähig sie ist, während sich Ihre Benutzeroberfläche im Laufe der Zeit verändert und weiterentwickelt.

Lassen Sie uns einen Service für unser theoretisches Auftragsabwicklungssystem erstellen, der versucht, proaktiv die richtige Lieferadresse für den Benutzer auszufüllen.

```
1 class OrderShippingAddressSubscriber
2   include Raix::ChatCompletion
3
4   attr_accessor :order
5
6   delegate :customer, to: :order
7
8   DIRECTIVE = "You are a smart order processing assistant. Given the
9   customer's order history, guess the most likely shipping address
10  for the current order."
11
12  def order_created(order)
13    return unless order.pending? && order.shipping_address.blank?
14
15    self.order = order
16
17    transcript.clear
18    transcript << { system: DIRECTIVE }
```

```
19     transcript << { user: "Order History: #{order_history.to_json}" }
20     transcript << { user: "Current Order: #{order.to_json}" }
21
22     response = chat_completion
23     apply_predicted_shipping_address(order, response)
24 end
25
26 private
27
28 def apply_predicted_shipping_address(order, response)
29   # extract the shipping address from the response...
30   # ...and assume there's some sort of live update of the address fields
31   order.update(shipping_address:)
32 end
33
34 def order_history
35   customer.orders.successful.limit(100).map do |order|
36     {
37       date: order.date,
38       description: order.description,
39       shipping_address: order.shipping_address
40     }
41   end
42 end
43 end
```

Dieses Beispiel ist sehr vereinfacht, sollte aber für die meisten Fälle funktionieren. Die Idee ist es, die KI auf die gleiche Weise eine Vermutung anstellen zu lassen wie ein Mensch es tun würde. Um zu verdeutlichen, wovon ich spreche, betrachten wir einige Beispieldaten:

1 Order History:

```
2 [  
3   {"date": "2024-01-03", "description": "garden soil mix",  
4     "shipping_address": "123 Country Lane, Rural Town"},  
5   {"date": "2024-01-15", "description": "hardcover fiction novels",  
6     "shipping_address": "456 City Apt, Metroville"},  
7   {"date": "2024-01-22", "description": "baby diapers", "shipping_address":  
8     "789 Suburb St, Quietville"},  
9   {"date": "2024-02-01", "description": "organic vegetables",  
10    "shipping_address": "123 Country Lane, Rural Town"},  
11  {"date": "2024-02-17", "description": "mystery thriller book set",  
12    "shipping_address": "456 City Apt, Metroville"},  
13  {"date": "2024-02-25", "description": "baby wipes",  
14    "shipping_address": "789 Suburb St, Quietville"},  
15  {"date": "2024-03-05", "description": "flower seeds",  
16    "shipping_address": "123 Country Lane, Rural Town"},  
17  {"date": "2024-03-20", "description": "biographies",  
18    "shipping_address": "456 City Apt, Metroville"},  
19  {"date": "2024-03-30", "description": "baby formula",  
20    "shipping_address": "789 Suburb St, Quietville"},  
21  {"date": "2024-04-12", "description": "lawn fertilizer",  
22    "shipping_address": "123 Country Lane, Rural Town"},  
23  {"date": "2024-04-22", "description": "science fiction novels",  
24    "shipping_address": "456 City Apt, Metroville"},  
25  {"date": "2024-05-02", "description": "infant toys",  
26    "shipping_address": "789 Suburb St, Quietville"},  
27  {"date": "2024-05-14", "description": "outdoor grill",  
28    "shipping_address": "123 Country Lane, Rural Town"},  
29  {"date": "2024-05-29", "description": "literary classics",  
30    "shipping_address": "456 City Apt, Metroville"},  
31  {"date": "2024-06-11", "description": "baby clothes",  
32    "shipping_address": "789 Suburb St, Quietville"},  
33  {"date": "2024-07-01", "description": "watering can",  
34    "shipping_address": "123 Country Lane, Rural Town"},  
35  {"date": "2024-07-18", "description": "non-fiction essays",  
36    "shipping_address": "456 City Apt, Metroville"},  
37  {"date": "2024-07-28", "description": "baby bath items",  
38    "shipping_address": "789 Suburb St, Quietville"},  
39  {"date": "2024-08-09", "description": "herb garden kit",  
40    "shipping_address": "123 Country Lane, Rural Town"},  
41  {"date": "2024-08-24", "description": "children's books",  
42    "shipping_address": "456 City Apt, Metroville"}]
```


43]

Ist dir das Muster in den Daten aufgefallen? Ich garantiere dir, das ist ein Kinderspiel für ein LLM. Zur Demonstration fragen wir GPT-4, was die wahrscheinlichste Lieferadresse für ein "Thermometer" ist.

```
1  From the order history you've provided, it looks like the purchases are
2  generally clustered into three main types based on the shipping addresses:
3
4  123 Country Lane, Rural Town - This address often orders garden and
5  outdoor-related items like soil mix, vegetables, flower seeds, lawn
6  fertilizer, an outdoor grill, a watering can, and an herb garden kit.
7
8  456 City Apt, Metroville - This address tends to order various types of
9  books, indicating an interest in reading across genres like fiction,
10  thrillers, biographies, science fiction, literary classics, non-fiction
11  essays, and children's books.
12
13  789 Suburb St, Quietville - The purchases for this address are consistently
14  baby-related products, such as diapers, wipes, baby formula, infant toys,
15  baby clothes, and baby bath items.
16
17  Given these patterns, a digital thermometer could potentially be a fit for
18  any of the addresses based on different needs: outdoor temperature tracking,
19  a tool for a passionate reader interested in science gadgets, or a health
20  tool for a baby. However, considering its common use in monitoring health,
21  especially for babies, it's most likely to be purchased by someone who's
22  already buying baby care products.
23
24  Therefore, the most likely shipping address for a digital thermometer, based
25  on the purchasing patterns, would be 789 Suburb St, Quietville.
```

Wenn Sie denken, dass es übertrieben ist, ein teures Modell wie GPT-4 für diese Aufgabe zu verwenden, haben Sie Recht! Ich habe denselben Prompt mit Mistral 7B Instruct getestet, und es lieferte die folgende Antwort mit 75 Token pro Sekunde, zu minimalen Kosten von 0,000218 USD.

```
1 Based on the given order history, it appears that the customer has received
2 orders for baby-related items (diapers, wipes, formula, clothes, bath items)
3 at the address "789 Suburb St, Quietville" and garden-related items (soil,
4 seeds, watering can, herb garden kit) at the address "123 Country Lane, Rural
5 Town". The customer has also received books (fiction, non-fiction, literary
6 classics, mystery-thriller, science fiction, biographies, children's books)
7 at the address "456 City Apt, Metroville".
8
9 Given that the new order is for a "thermometer", which doesn't seem to fit
10 into any of the categories that the customer has previously ordered, it's
11 hard to accurately predict the shipping address based solely on the order
12 history. However, given the 50%-50% split between baby-related and
13 garden-related items, it could somewhat lean towards the Baby-related items
14 address ("789 Suburb St, Quietville"). But remember, this is an assumption
15 and cannot be definitively confirmed without more context or information.
```

Ist der Aufwand und die Kosten dieser Technik es wert, um ein Checkout-Erlebnis magischer zu gestalten? Für viele Online-Händler auf jeden Fall. Und wie es aussieht, werden die Kosten für KI-Berechnungen nur noch weiter sinken, besonders bei den Anbietern von Open-Source-Modell-Hosting, die sich in einem Preiskampf befinden.



Verwenden Sie ein [Prompt Template](#) und [StructuredIO](#) zusammen mit [Response Fencing](#), um diese Art von Chat-Completion zu optimieren.

Adaptive Feldanordnung

Die Reihenfolge, in der Formularfelder präsentiert werden, kann einen erheblichen Einfluss auf das Benutzererlebnis und die Abschlussquoten haben. Mit GenUI können Sie die Feldanordnung dynamisch an den Kontext des Benutzers und die Wichtigkeit jedes Feldes anpassen. Wenn ein Benutzer beispielsweise ein Registrierungsformular für eine Fitness-App ausfüllt, könnte das Formular Felder priorisieren, die mit seinen Fitnesszielen und Präferenzen zusammenhängen, wodurch der Prozess relevanter und ansprechender wird.

Personalisierte Microcopy

Der Anleitungstext, Fehlermeldungen und andere Microcopy in Verbindung mit Formularen können ebenfalls mit GenUI personalisiert werden. Anstatt generische Fehlermeldungen wie “Ungültige E-Mail-Adresse” anzuzeigen, können Sie hilfreichere und kontextbezogene Nachrichten generieren, wie zum Beispiel “Bitte geben Sie eine gültige E-Mail-Adresse ein, um Ihre Bestellbestätigung zu erhalten.” Diese personalisierten Anpassungen können das Formularerlebnis benutzerfreundlicher und weniger frustrierend gestalten.

Personalisierte Validierung

Ähnlich wie bei der Personalisierten Microcopy könnten Sie KI nutzen, um das Formular auf eine scheinbar magische Weise zu validieren. Stellen Sie sich vor, eine KI validiert ein Benutzerprofil-Formular und sucht nach möglichen Fehlern auf einer *semantischen* Ebene.

Create your account

Full name

Obie Fernandez

Email

obiefernandez@gmail.com



Did you mean obiefernandez@gmail.com? [Yes, update.](#)

Country ⓘ

 United States



Password

.....



✓ Nice work. This is an excellent password.

Abbildung 9. Können Sie die semantische Validierung erkennen?

Progressive Disclosure

GenUI kann intelligent bestimmen, welche Formularfelder basierend auf dem Benutzerkontext essentiell sind und zusätzliche Felder nach Bedarf schrittweise einblenden. Diese Progressive-Disclosure-Technik hilft dabei, die kognitive Belastung zu reduzieren und macht den Prozess des Formularausfüllens überschaubarer. Wenn

sich ein Benutzer beispielsweise für ein Basis-Abonnement anmeldet, kann das Formular zunächst nur die wesentlichen Felder anzeigen, und während der Benutzer fortschreitet oder bestimmte Optionen auswählt, können zusätzliche relevante Felder dynamisch eingeführt werden.

Kontextbezogener Erklärungstext

Tooltips werden häufig verwendet, um Benutzern zusätzliche Informationen oder Anleitungen zu geben, wenn sie über bestimmte Elemente hovern oder mit ihnen interagieren. Mit einem “Kontextbezogenen Content-Generierungs”-Ansatz können Sie Tooltips erstellen, die sich an den Kontext des Benutzers anpassen und relevante Informationen liefern. Wenn ein Benutzer beispielsweise eine komplexe Funktion erkundet, kann der Tooltip personalisierte Tipps oder Beispiele basierend auf seinen vorherigen Interaktionen oder seinem Kenntnisstand anbieten.

Erklärungstexte, wie Anweisungen, Beschreibungen oder Hilfemeldungen, können dynamisch basierend auf dem Benutzerkontext generiert werden. Anstatt generische Erklärungen zu präsentieren, können Sie LLMs verwenden, um Text zu generieren, der auf die spezifischen Bedürfnisse oder Fragen des Benutzers zugeschnitten ist. Wenn ein Benutzer beispielsweise bei einem bestimmten Schritt in einem Prozess Schwierigkeiten hat, kann der Erklärungstext personalisierte Anleitungen oder Fehlerbehebungstipps bereitstellen.

Microcopy bezieht sich auf die kleinen Textelemente, die Benutzer durch Ihre Anwendung führen, wie Schaltflächenbeschriftungen, Fehlermeldungen oder Bestätigungsaufforderungen. Durch die Anwendung des [Kontextbezogenen Content-Generierungs](#)-Ansatzes auf Microcopy können Sie eine adaptive Benutzeroberfläche erstellen, die auf die Aktionen des Benutzers reagiert und relevanten und hilfreichen Text bereitstellt. Wenn ein Benutzer beispielsweise im Begriff ist, eine kritische Aktion durchzuführen, kann die Bestätigungsaufforderung dynamisch generiert werden, um eine klare und personalisierte Nachricht bereitzustellen.

Personalisierte Erklärungstexte und Tooltips können den Onboarding-Prozess für neue Benutzer erheblich verbessern. Durch die Bereitstellung kontextspezifischer Anleitungen und Beispiele können Sie Benutzern helfen, die Anwendung schnell zu verstehen und zu navigieren, wodurch die Lernkurve reduziert und die Akzeptanz erhöht wird.

Dynamische und kontextbezogene Chrome-Elemente können die Anwendung auch intuitiver und ansprechender gestalten. Benutzer sind eher geneigt, mit Funktionen zu interagieren und diese zu erkunden, wenn der begleitende Text auf ihre spezifischen Bedürfnisse und Interessen zugeschnitten ist.

Bisher haben wir Ideen zur Verbesserung bestehender UI-Paradigmen mit KI behandelt, aber wie steht es damit, die Art und Weise, wie Benutzeroberflächen gestaltet und implementiert werden, auf radikalere Weise zu überdenken?

Definition der Generativen UI

Im Gegensatz zum traditionellen UI-Design, bei dem Designer feste, statische Schnittstellen erstellen, deutet GenUI auf eine Zukunft hin, in der unsere Software flexible, personalisierte Erlebnisse bietet, die sich in Echtzeit entwickeln und anpassen können. Jedes Mal, wenn wir eine KI-gesteuerte Konversationsschnittstelle nutzen, lassen wir die KI sich an die speziellen Bedürfnisse des Benutzers anpassen. GenUI geht noch einen Schritt weiter, indem es diesen Grad der Anpassungsfähigkeit auf die *visuelle* Schnittstelle der Software anwendet.

Der Grund, warum es heute möglich ist, mit GenUI-Ideen zu experimentieren, liegt darin, dass große Sprachmodelle bereits Programmierung verstehen und ihr Basiswissen UI-Technologien und Frameworks umfasst. Die Frage ist also, ob große Sprachmodelle zur Generierung von UI-Elementen wie Text, Bildern, Layouts und sogar ganzen

Schnittstellen genutzt werden können, die auf jeden einzelnen Benutzer zugeschnitten sind. Das Modell könnte angewiesen werden, verschiedene Faktoren zu berücksichtigen, wie etwa frühere Interaktionen des Benutzers, angegebene Präferenzen, demografische Informationen und den aktuellen Nutzungskontext, um hochgradig personalisierte und relevante Schnittstellen zu erstellen.

GenUI unterscheidet sich in mehreren wesentlichen Punkten vom traditionellen Design von Benutzeroberflächen:

1. **Dynamisch und Adaptiv:** Traditionelles UI-Design beinhaltet die Erstellung fester, statischer Schnittstellen, die für alle Benutzer gleich bleiben. Im Gegensatz dazu ermöglicht GenUI Schnittstellen, die sich dynamisch an Benutzerbedürfnisse und Kontext anpassen können. Das bedeutet, dass dieselbe Anwendung verschiedenen Benutzern unterschiedliche Schnittstellen präsentieren kann oder sogar demselben Benutzer in unterschiedlichen Situationen.
2. **Personalisierung im großen Maßstab:** Beim traditionellen Design ist die Erstellung personalisierter Erlebnisse für jeden Benutzer oft aufgrund des Zeit- und Ressourcenaufwands unpraktisch. GenUI hingegen ermöglicht Personalisierung im großen Maßstab. Durch den Einsatz von KI können Designer Schnittstellen erstellen, die sich automatisch an die einzigartigen Bedürfnisse und Präferenzen jedes Benutzers anpassen, ohne manuell separate Schnittstellen für jedes Benutzersegment entwickeln zu müssen.
3. **Fokus auf Ergebnisse:** Traditionelles UI-Design konzentriert sich oft auf die Erstellung visuell ansprechender und funktionaler Schnittstellen. Während diese Aspekte auch bei GenUI wichtig bleiben, verlagert sich der Hauptfokus auf das Erreichen gewünschter Benutzerergebnisse. GenUI zielt darauf ab, Schnittstellen zu erstellen, die für die spezifischen Ziele und Aufgaben jedes Benutzers optimiert sind, wobei Benutzerfreundlichkeit und Effektivität Vorrang vor rein ästhetischen Überlegungen haben.
4. **Kontinuierliches Lernen und Verbessern:** GenUI-Systeme können basierend auf Benutzerinteraktionen und Feedback kontinuierlich lernen und sich verbessern.

Während Benutzer mit den generierten Schnittstellen interagieren, können die KI-Modelle Daten über Benutzerverhalten, Präferenzen und Ergebnisse sammeln und diese Informationen nutzen, um zukünftige Schnittstellengenerationen zu verfeinern und zu optimieren. Dieser iterative Lernprozess ermöglicht es GenUI-Systemen, im Laufe der Zeit immer effektiver bei der Erfüllung von Benutzerbedürfnissen zu werden.

Es ist wichtig zu beachten, dass GenUI nicht dasselbe ist wie KI-unterstützte Designwerkzeuge, wie solche, die Vorschläge machen oder bestimmte Designaufgaben automatisieren. Während diese Werkzeuge bei der Optimierung des Designprozesses hilfreich sein können, sind sie immer noch auf Designer angewiesen, die endgültige Entscheidungen treffen und statische Schnittstellen erstellen. Bei GenUI hingegen übernimmt das KI-System eine aktivere Rolle bei der tatsächlichen Generierung und Anpassung von Schnittstellen basierend auf Benutzerdaten und Kontext.

GenUI stellt eine bedeutende Veränderung in unserem Ansatz zum Design von Benutzeroberflächen dar, weg von Einheitslösungen und hin zu hochgradig personalisierten, adaptiven Erlebnissen. Durch die Nutzung der Kraft der KI hat GenUI das Potenzial, die Art und Weise zu revolutionieren, wie wir mit digitalen Produkten und Dienstleistungen interagieren, indem es Schnittstellen schafft, die für jeden einzelnen Benutzer intuitiver, ansprechender und effektiver sind.

Beispiel

Um das Konzept von GenUI zu veranschaulichen, betrachten wir eine hypothetische Fitness-Anwendung namens "FitAI". Diese App zielt darauf ab, personalisierte Trainingspläne und Ernährungsberatung für Benutzer basierend auf ihren individuellen Zielen, Fitnessleveln und Präferenzen bereitzustellen.

In einem traditionellen UI-Design-Ansatz hätte FitAI möglicherweise einen festen Satz von Bildschirmen und Elementen, die für alle Benutzer gleich sind. Mit GenUI könnte

sich die Schnittstelle der App jedoch dynamisch an die einzigartigen Bedürfnisse und den Kontext jedes Benutzers anpassen.

Dieser Ansatz ist für 2024 schwer vorstellbar und könnte möglicherweise nicht einmal einen angemessenen ROI haben, aber er ist möglich.

So könnte es funktionieren:

1. Onboarding:

- Anstelle eines standardisierten Fragebogens verwendet FitAI eine Konversations-KI, um Informationen über die Ziele, das aktuelle Fitnesslevel und die Präferenzen des Benutzers zu sammeln.
- Basierend auf dieser ersten Interaktion generiert die KI ein personalisiertes Dashboard-Layout, das die für die Ziele des Benutzers relevantesten Funktionen und Informationen hervorhebt.
- Die aktuelle KI-Technologie könnte über eine Auswahl von Bildschirmkomponenten verfügen, die sie zur Zusammenstellung des personalisierten Dashboards verwenden kann.
- Zukünftige KI-Technologie könnte die Rolle eines erfahrenen UI-Designers übernehmen und das Dashboard tatsächlich *von Grund auf* neu erstellen.

2. Trainingsplaner:

- Die Trainingsplaner-Schnittstelle wird von der KI speziell an das Erfahrungsniveau und die verfügbare Ausrüstung des Nutzers angepasst.
- Für einen Anfänger ohne Ausrüstung könnte sie einfache Körpergewichtsübungen mit detaillierten Anleitungen und Videos anzeigen.
- Für einen fortgeschrittenen Nutzer mit Zugang zu einem Fitnessstudio könnte sie komplexere Routinen mit weniger Erklärungen darstellen.
- Der Inhalt des Trainingsplaners wird nicht einfach aus einer großen Übergruppe gefiltert. Er kann *spontan* aus einer Wissensbasis generiert werden, die mit Kontext abgefragt wird, der alles über den Nutzer Bekannte einschließt.

3. Fortschrittsverfolgung:

- Die Fortschrittsverfolgungsschnittstelle entwickelt sich basierend auf den Zielen und Nutzungsmustern des Anwenders.
- Wenn ein Nutzer hauptsächlich auf Gewichtsverlust fokussiert ist, könnte die Oberfläche prominent eine Gewichtsentwicklungsgrafik und Kalorienverbrauchsstatistiken anzeigen.
- Für einen Nutzer, der Muskeln aufbaut, könnte sie Kraftzuwächse und Körperzusammensetzungsänderungen hervorheben.
- Die KI kann diesen Teil der Anwendung an den tatsächlichen Fortschritt des Nutzers anpassen. Wenn der Fortschritt für eine Zeit lang stoppt, kann die App in einen Modus wechseln, in dem sie versucht, den Nutzer dazu zu bringen, die Gründe für den Rückschlag zu offenbaren, um sie zu beheben.

4. Ernährungsberatung:

- Der Ernährungsbereich passt sich an die Ernährungspräferenzen und -einschränkungen des Nutzers an.
- Für einen veganen Nutzer könnte er pflanzliche Mahlzeitenvorschläge und Proteinquellen anzeigen.
- Für einen Nutzer mit Glutenunverträglichkeit würde er automatisch glutenhaltige Lebensmittel aus den Empfehlungen herausfiltern.
- Auch hier wird der Inhalt nicht aus einer riesigen Übergruppe von Mahlzeitendaten gezogen, die für alle Nutzer gilt, sondern aus einer Wissensbasis synthetisiert, die Informationen enthält, die basierend auf der spezifischen Situation und den Einschränkungen des Nutzers anpassbar sind.
- Zum Beispiel werden Rezepte mit Zutatenspezifikationen generiert, die dem sich ständig ändernden Kalorienbedarf des Nutzers entsprechen, während sich sein Fitnesslevel und seine Körperwerte entwickeln.

5. Motivationselemente:

- Die motivierenden Inhalte und Benachrichtigungen der App werden basierend auf dem Persönlichkeitstyp des Nutzers und seiner Reaktion auf verschiedene Motivationsstrategien personalisiert.
- Einige Nutzer erhalten möglicherweise ermutigende Nachrichten, während andere eher datengesteuerte Rückmeldungen bekommen.

In diesem Beispiel ermöglicht GenUI FitAI, eine hochgradig personalisierte Erfahrung für jeden Nutzer zu schaffen, was potenziell das Engagement, die Zufriedenheit und die Wahrscheinlichkeit der Erreichung von Fitnesszielen erhöht. Die Schnittstellenelemente, Inhalte und sogar die “Persönlichkeit” der App passen sich an, um den individuellen Bedürfnissen und Präferenzen jedes Nutzers optimal zu dienen.

Der Wandel zum ergebnisorientierten Design

GenUI stellt einen fundamentalen Wandel im Ansatz des Benutzeroberflächen-Designs dar, der sich von einem Fokus auf die Erstellung spezifischer Schnittstellenelemente zu einem ganzheitlicheren, ergebnisorientierten Ansatz bewegt. Dieser Wandel hat mehrere wichtige Auswirkungen:

1. Fokus auf Nutzerziele:

- Designer müssen intensiver über Nutzerziele und gewünschte Ergebnisse nachdenken, anstatt über spezifische Schnittstellenkomponenten.
- Der Schwerpunkt wird darauf liegen, Systeme zu schaffen, die Schnittstellen generieren können, die Nutzern effizient und effektiv bei der Erreichung ihrer Ziele helfen.
- Neue UI-Frameworks werden entstehen, die KI-basierten Designern die Werkzeuge geben, die sie benötigen, um Nutzererfahrungen *spontan* und *von Grund auf* zu generieren, anstatt auf vordefinierte Bildschirmspezifikationen zu setzen.

2. Veränderte Rolle der Designer:

- Designer werden sich von der Erstellung fester Layouts hin zur Definition von Regeln, Einschränkungen und Richtlinien entwickeln, denen KI-Systeme bei der Generierung von Schnittstellen folgen.
- Sie müssen Fähigkeiten in Bereichen wie Datenanalyse, KI-Prompt-Engineering und Systemdenken entwickeln, um GenUI-Systeme effektiv zu steuern.

3. Bedeutung der Nutzerforschung:

- Nutzerforschung wird in einem GenUI-Kontext noch wichtiger, da Designer nicht nur Nutzerpräferenzen verstehen müssen, sondern auch, wie sich diese Präferenzen und Bedürfnisse in verschiedenen Kontexten ändern.
- Kontinuierliche Nutzertests und Feedback-Schleifen werden essentiell sein, um die Fähigkeit der KI zur Generierung effektiver Schnittstellen zu verfeinern und zu verbessern.

4. Design für Variabilität:

- Anstatt eine einzige “perfekte” Schnittstelle zu erstellen, müssen Designer mehrere mögliche Variationen berücksichtigen und sicherstellen, dass das System angemessene Schnittstellen für verschiedene Nutzerbedürfnisse generieren kann.
- Dies beinhaltet das Design für Randfälle und die Sicherstellung, dass die generierten Schnittstellen über verschiedene Konfigurationen hinweg Benutzerfreundlichkeit und Zugänglichkeit bewahren.
- Produktdifferenzierung erhält neue Dimensionen, die unterschiedliche Perspektiven auf Nutzerpsychologie und die Nutzung einzigartiger Datensätze und Wissensbasen einbeziehen, die Wettbewerbern nicht zur Verfügung stehen.

Herausforderungen und Überlegungen

Während GenUI spannende Möglichkeiten bietet, bringt es auch mehrere Herausforderungen und Überlegungen mit sich:

1. Technische Einschränkungen:

- Die aktuelle KI-Technologie hat trotz ihrer Fortschrittlichkeit noch Einschränkungen beim Verständnis komplexer Nutzerabsichten und der Generierung wirklich kontextbewusster Schnittstellen.
- Leistungsprobleme im Zusammenhang mit der Echtzeit-Generierung von Schnittstellenelementen, besonders auf weniger leistungsfähigen Geräten.

2. Datenanforderungen:

- Je nach Anwendungsfall könnten effektive GenUI-Systeme erhebliche Mengen an Nutzerdaten benötigen, um personalisierte Benutzeroberflächen zu generieren.
- Die Herausforderungen bei der ethischen Beschaffung authentischer Nutzerdaten werfen Bedenken hinsichtlich Datenschutz und Sicherheit sowie möglicher Verzerrungen in den Daten auf, die zum Training von GenUI-Modellen verwendet werden.

3. Benutzbarkeit und Konsistenz:

- Zumindest bis sich die Praxis weitgehend durchgesetzt hat, könnte eine Anwendung mit sich ständig ändernden Benutzeroberflächen zu Benutzbarkeitsproblemen führen, da Nutzer Schwierigkeiten haben könnten, vertraute Elemente zu finden oder effizient zu navigieren.
- Es wird entscheidend sein, eine Balance zwischen Personalisierung und der Aufrechterhaltung einer konsistenten, erlernbaren Benutzeroberfläche zu finden.

4. Übermäßige Abhängigkeit von KI:

- Es besteht das Risiko, Design-Entscheidungen zu sehr an KI-Systeme zu delegieren, was zu uninspirierten, problematischen oder einfach fehlerhaften Schnittstellenlösungen führen könnte.
- Menschliche Aufsicht und die Möglichkeit, KI-generierte Designs zu übersteuern, werden in absehbarer Zukunft wichtig bleiben.

5. Bedenken zur Barrierefreiheit:

- Die Sicherstellung, dass dynamisch generierte Benutzeroberflächen für Nutzer mit Behinderungen zugänglich bleiben, stellt völlig neue Herausforderungen dar, was angesichts des mangelhaften Niveaus der Barrierefreiheit in typischen Systemen besorgniserregend ist.
- Andererseits könnten KI-Designer mit *eingebauter* Berücksichtigung der Barrierefreiheit implementiert werden und Fähigkeiten zur spontanen Erstellung barrierefreier Benutzeroberflächen entwickeln, ähnlich wie sie UI für nicht-beeinträchtigte Nutzer erstellen.
- In jedem Fall sollten GenUI-Systeme mit robusten Richtlinien und Testprozessen für Barrierefreiheit entwickelt werden.

6. Nutzervertrauen und Transparenz:

- Nutzer könnten sich unwohl fühlen mit Benutzeroberflächen, die “zu viel” über sie zu wissen scheinen oder sich auf unverständliche Weise ändern.
- Es wird wichtig sein, Transparenz darüber zu schaffen, wie und warum Benutzeroberflächen personalisiert werden, um Nutzervertrauen aufzubauen.

Zukunftsausblick und Chancen

Die Zukunft von Generative UI (GenUI) verspricht eine revolutionäre Veränderung in der Art und Weise, wie wir mit digitalen Produkten und Dienstleistungen interagieren. Während sich diese Technologie weiterentwickelt, können wir einen grundlegenden Wandel in der Art und Weise erwarten, wie Benutzeroberflächen gestaltet, implementiert und erlebt werden. Ich denke, GenUI ist das Phänomen, das unsere Software endlich in den Bereich dessen bringen wird, was heute als Science-Fiction gilt.

Eine der spannendsten Aussichten von GenUI ist sein Potenzial, Barrierefreiheit in einem Ausmaß zu verbessern, das über die bloße Sicherstellung hinausgeht, dass Menschen mit schweren Behinderungen nicht völlig von der Nutzung Ihrer Software ausgeschlossen werden. Durch die automatische Anpassung von Benutzeroberflächen an individuelle Nutzerbedürfnisse könnte GenUI digitale Erlebnisse inklusiver machen als je zuvor. Stellen Sie sich Benutzeroberflächen vor, die sich nahtlos anpassen, um größeren Text für jüngere oder sehbehinderte Nutzer oder vereinfachte Layouts für Menschen mit kognitiven Einschränkungen bereitzustellen, ohne dass manuelle Konfigurationen oder separate "barrierefreie" Versionen von Anwendungen erforderlich sind.

Die Personalisierungsmöglichkeiten von GenUI werden voraussichtlich zu einer erhöhten Nutzereinbindung, -zufriedenheit und -loyalität bei einer breiten Palette digitaler Produkte führen. Wenn Benutzeroberflächen besser auf individuelle Präferenzen und Verhaltensweisen abgestimmt sind, werden Nutzer digitale Erlebnisse intuitiver und angenehmer finden, was potenziell zu tieferen und bedeutungsvolleren Interaktionen mit Technologie führt.

GenUI hat auch das Potenzial, den Einarbeitungsprozess für neue Nutzer zu transformieren. Durch die Schaffung intuitiver, personalisierter Erstnutzererfahrungen, die sich schnell an das Expertenniveau jedes Nutzers anpassen, könnte GenUI die

Lernkurve für neue Anwendungen deutlich reduzieren. Dies könnte zu schnelleren Adoptionsraten und erhöhtem Nutzervertrauen bei der Erkundung neuer Funktionen führen.

Eine weitere spannende Möglichkeit ist die Fähigkeit von GenUI, eine konsistente Nutzererfahrung über verschiedene Geräte und Plattformen hinweg zu gewährleisten und dabei für jeden spezifischen Nutzungskontext zu optimieren. Dies könnte die langjährige Herausforderung lösen, kohärente Erlebnisse in einer zunehmend fragmentierten Gerätelandschaft zu bieten, von Smartphones und Tablets bis hin zu Desktop-Computern und aufkommenden Technologien wie Augmented-Reality-Brillen.

Die datengetriebene Natur von GenUI eröffnet Möglichkeiten für schnelle Iteration und Verbesserung im UI-Design. Durch die Sammlung von Echtzeitdaten darüber, wie Nutzer mit generierten Benutzeroberflächen interagieren, können Designer und Entwickler beispiellose Einblicke in Nutzerverhalten und -präferenzen gewinnen. Diese Feedback-Schleife könnte zu kontinuierlichen Verbesserungen im UI-Design führen, die von tatsächlichen Nutzungsmustern statt von Annahmen oder begrenzten Nutzertests getrieben werden.

Um sich auf diesen Wandel vorzubereiten, müssen Designer ihre Fähigkeiten und Denkweisen weiterentwickeln. Der Fokus wird sich von der Erstellung fester Layouts hin zur Entwicklung umfassender Design-Systeme und Richtlinien verschieben, die die KI-gesteuerte Schnittstellengenerierung informieren können. Designer werden ein tiefes Verständnis von Datenanalyse, KI-Technologien und Systemdenken entwickeln müssen, um GenUI-Systeme effektiv zu steuern.

Darüber hinaus werden Designer, da GenUI die Grenzen zwischen Design und Technologie verwischt, enger mit Entwicklern und Data Scientists zusammenarbeiten müssen. Dieser interdisziplinäre Ansatz wird entscheidend sein, um GenUI-Systeme zu schaffen, die nicht nur visuell ansprechend und benutzerfreundlich, sondern auch technisch robust und ethisch einwandfrei sind.

Die ethischen Implikationen von GenUI werden mit der Weiterentwicklung der Technologie ebenfalls in den Vordergrund rücken. Designer werden eine entscheidende Rolle bei der Entwicklung von Rahmenwerken für den verantwortungsvollen Einsatz von KI im Schnittstellendesign spielen und dabei sicherstellen, dass Personalisierung die Benutzererfahrung verbessert, ohne dabei die Privatsphäre zu gefährden oder das Nutzerverhalten auf unethische Weise zu manipulieren.

Mit Blick auf die Zukunft bietet GenUI sowohl spannende Möglichkeiten als auch erhebliche Herausforderungen. Die Technologie hat das Potenzial, intuitivere, effizientere und zufriedenstellendere digitale Erfahrungen für Nutzer weltweit zu schaffen. Während Designer sich anpassen und neue Fähigkeiten erwerben müssen, bietet sich auch eine beispiellose Gelegenheit, die Zukunft der Mensch-Computer-Interaktion auf tiefgreifende und bedeutungsvolle Weise zu gestalten. Der Weg zu vollständig entwickelten GenUI-Systemen wird zweifellos komplex sein, aber die potenziellen Vorteile im Hinblick auf verbesserte Benutzererfahrungen und digitale Barrierefreiheit machen es zu einer Zukunft, für die es sich zu streben lohnt.

Intelligente Workflow-Orchestrierung



Im Bereich der Anwendungsentwicklung spielen Workflows eine entscheidende Rolle bei der Definition, wie Aufgaben, Prozesse und Benutzerinteraktionen strukturiert und ausgeführt werden. Mit zunehmender Komplexität von Anwendungen und steigenden Benutzererwartungen wird der Bedarf an intelligenter und adaptiver Workflow-Orchestrierung immer deutlicher.

Der Ansatz der “Intelligenten Workflow-Orchestrierung” konzentriert sich darauf, KI-Komponenten zu nutzen, um komplexe Workflows innerhalb von Anwendungen dynamisch zu orchestrieren und zu optimieren. Das Ziel ist es, Anwendungen zu erstellen, die effizienter, reaktionsschneller und anpassungsfähiger an Echtzeitdaten und Kontext sind.

In diesem Kapitel werden wir die wichtigsten Prinzipien und Muster erkunden, die dem Ansatz der intelligenten Workflow-Orchestrierung zugrunde liegen. Wir werden betrachten, wie KI eingesetzt werden kann, um Aufgaben intelligent zu leiten, Entscheidungsfindung zu automatisieren und Workflows dynamisch an verschiedene Faktoren wie Benutzerverhalten, Systemleistung und Geschäftsregeln anzupassen. Anhand praktischer Beispiele und realer Szenarien werden wir das transformative Potenzial von KI bei der Optimierung von Anwendungs-Workflows demonstrieren.

Ob Sie Unternehmensanwendungen mit komplexen Geschäftsprozessen oder kundenorientierte Anwendungen mit dynamischen Benutzerreisen entwickeln, die in diesem Kapitel diskutierten Muster und Techniken werden Sie mit dem Wissen und den Werkzeugen ausstatten, um intelligente und effiziente Workflows zu erstellen, die die gesamte Benutzererfahrung verbessern und geschäftlichen Mehrwert schaffen.

Geschäftlicher Bedarf

Traditionelle Ansätze zum Workflow-Management basieren oft auf vordefinierten Regeln und statischen Entscheidungsbäumen, die starr und unflexibel sein können und nicht mit der dynamischen Natur moderner Anwendungen Schritt halten können.

Betrachten Sie ein Szenario, in dem eine E-Commerce-Anwendung einen komplexen Auftragsabwicklungsprozess handhaben muss. Der Workflow kann mehrere Schritte umfassen, wie Auftragsvalidierung, Bestandsprüfung, Zahlungsabwicklung, Versand und Kundenbenachrichtigungen. Jeder Schritt kann eigene Regeln, Abhängigkeiten, externe Integrationen und Mechanismen zur Ausnahmebehandlung haben. Die manuelle Verwaltung eines solchen Workflows oder die Verwendung fest programmierter Logik kann schnell umständlich, fehleranfällig und schwer zu warten werden.

Darüber hinaus muss sich der Workflow möglicherweise mit zunehmender Skalierung der Anwendung und wachsender Anzahl gleichzeitiger Benutzer auf Basis von

Echtzeitdaten und Systemleistung anpassen und optimieren. Beispielsweise muss die Anwendung während Spitzenverkehrszeiten den Workflow möglicherweise dynamisch anpassen, um bestimmte Aufgaben zu priorisieren, Ressourcen effizient zuzuweisen und eine reibungslose Benutzererfahrung sicherzustellen.

Hier kommt der Ansatz der "Intelligenten Workflow-Orchestrierung" ins Spiel. Durch den Einsatz von KI-Komponenten können Entwickler Workflows erstellen, die intelligent, adaptiv und selbstoptimierend sind. KI kann große Mengen an Daten analysieren, aus vergangenen Erfahrungen lernen und in Echtzeit fundierte Entscheidungen treffen, um den Workflow effektiv zu orchestrieren.

Wichtige Vorteile

1. **Gesteigerte Effizienz:** KI kann die Aufgabenzuweisung, Ressourcennutzung und Workflow-Ausführung optimieren, was zu schnelleren Verarbeitungszeiten und verbesserter Gesamteffizienz führt.
2. **Anpassungsfähigkeit:** KI-gesteuerte Workflows können sich dynamisch an veränderte Bedingungen anpassen, wie Schwankungen in der Benutzernachfrage, Systemleistung oder geschäftlichen Anforderungen, und stellen sicher, dass die Anwendung reaktionsfähig und widerstandsfähig bleibt.
3. **Automatisierte Entscheidungsfindung:** KI kann komplexe Entscheidungsprozesse innerhalb des Workflows automatisieren, wodurch manuelle Eingriffe reduziert und das Risiko menschlicher Fehler minimiert werden.
4. **Personalisierung:** KI kann Benutzerverhalten, Präferenzen und Kontext analysieren, um den Workflow zu personalisieren und maßgeschneiderte Erlebnisse für einzelne Benutzer zu liefern.
5. **Skalierbarkeit:** KI-gesteuerte Workflows können nahtlos skalieren, um steigende Datenmengen und Benutzerinteraktionen zu bewältigen, ohne die Leistung oder Zuverlässigkeit zu beeinträchtigen.

In den folgenden Abschnitten werden wir die wichtigsten Muster und Techniken erkunden, die die Implementierung intelligenter Workflows ermöglichen, und reale Beispiele aufzeigen, wie KI das Workflow-Management in modernen Anwendungen transformiert.

Wichtige Muster

Um intelligente Workflow-Orchestrierung in Anwendungen zu implementieren, können Entwickler verschiedene wichtige Muster nutzen, die die Kraft der KI nutzen. Diese Muster bieten einen strukturierten Ansatz für das Design und Management von Workflows und ermöglichen es Anwendungen, sich anzupassen, zu optimieren und Prozesse basierend auf Echtzeitdaten und Kontext zu automatisieren. Lassen Sie uns einige der grundlegenden Muster in der intelligenten Workflow-Orchestrierung erkunden.

Dynamische Aufgabenweiterleitung

Dieses Muster beinhaltet die Verwendung von KI zur intelligenten Weiterleitung von Aufgaben innerhalb eines Workflows basierend auf verschiedenen Faktoren wie Aufgabenpriorität, Ressourcenverfügbarkeit und Systemleistung. KI-Algorithmen können die Eigenschaften jeder Aufgabe analysieren, den aktuellen Systemzustand berücksichtigen und fundierte Entscheidungen treffen, um Aufgaben den am besten geeigneten Ressourcen oder Verarbeitungspfaden zuzuweisen. Die dynamische Aufgabenweiterleitung stellt sicher, dass Aufgaben effizient verteilt und ausgeführt werden, wodurch die Gesamtleistung des Workflows optimiert wird.

```
1  class TaskRouter
2    include Raix::ChatCompletion
3    include Raix::FunctionDispatch
4
5    attr_accessor :task
6
7    # list of functions that can be called by the AI entirely at its
8    # discretion depending on the task received
9
10   function :analyze_task_priority do
11     TaskPriorityAnalyzer.perform(task)
12   end
13
14   function :check_resource_availability, # ...
15   function :assess_system_performance, # ...
16   function :assign_task_to_resource, # ...
17
18   DIRECTIVE = "You are a task router, responsible for intelligently
19     assigning tasks to available resources based on priority, resource
20     availability, and system performance..."
21
22   def initialize(task)
23     self.task = task
24     transcript << { system: DIRECTIVE }
25     transcript << { user: task.to_json }
26   end
27
28   def perform
29     while task.unassigned?
30       chat_completion
31
32       # todo: add max loop counter and break
33     end
34
35     # capture the transcript for later analysis
36     task.update(routing_transcript: transcript)
37   end
38 end
```

Beachten Sie die Schleife, die durch den while-Ausdruck in Zeile 29 erstellt wird, die die KI so lange auffordert, bis die Aufgabe zugewiesen ist. In Zeile 35 speichern wir das

Protokoll der Aufgabe für spätere Analysen und Debugging, falls dies erforderlich wird.

Kontextbezogene Entscheidungsfindung

Sie können sehr ähnlichen Code verwenden, um kontextbezogene Entscheidungen innerhalb eines Workflows zu treffen. Durch die Analyse relevanter Datenpunkte wie Benutzerpräferenzen, historische Muster und Echtzeitdaten können KI-Komponenten den am besten geeigneten Handlungsablauf an jedem Entscheidungspunkt im Workflow bestimmen. Passen Sie das Verhalten Ihres Workflows basierend auf dem spezifischen Kontext jedes Benutzers oder Szenarios an und bieten Sie personalisierte und optimierte Erfahrungen.

Adaptive Workflow-Zusammenstellung

Dieses Muster konzentriert sich auf die dynamische Zusammenstellung und Anpassung von Workflows basierend auf sich ändernden Anforderungen oder Bedingungen. KI kann den aktuellen Zustand des Workflows analysieren, Engpässe oder Ineffizienzen identifizieren und die Workflow-Struktur automatisch modifizieren, um die Leistung zu optimieren. Die adaptive Workflow-Zusammenstellung ermöglicht es Anwendungen, sich kontinuierlich weiterzuentwickeln und ihre Prozesse zu verbessern, ohne manuelle Eingriffe zu erfordern.

Ausnahmebehandlung und Wiederherstellung

Ausnahmebehandlung und Wiederherstellung sind kritische Aspekte der intelligenten Workflow-Orchestrierung. Bei der Arbeit mit KI-Komponenten und komplexen Workflows ist es wichtig, Ausnahmen vorauszusehen und elegant zu behandeln, um die Stabilität und Zuverlässigkeit des Systems sicherzustellen.

Hier sind einige wichtige Überlegungen und Techniken für die Ausnahmebehandlung und Wiederherstellung in intelligenten Workflows:

1. **Ausnahmeweiterleitung:** Implementieren Sie einen einheitlichen Ansatz für die Weiterleitung von Ausnahmen über Workflow-Komponenten hinweg. Wenn eine Ausnahme innerhalb einer Komponente auftritt, sollte sie erfasst, protokolliert und an den Orchestrator oder eine separate Komponente weitergeleitet werden, die für die Behandlung von Ausnahmen zuständig ist. Die Idee ist, die Ausnahmebehandlung zu zentralisieren und zu verhindern, dass Ausnahmen stillschweigend verschluckt werden, sowie Möglichkeiten für [Intelligente Fehlerbehandlung](#) zu eröffnen.
2. **Wiederholungsmechanismen:** Wiederholungsmechanismen helfen dabei, die Widerstandsfähigkeit des Workflows zu verbessern und vorübergehende Ausfälle elegant zu behandeln. Implementieren Sie unbedingt Wiederholungsmechanismen für vorübergehende oder behebbare Ausnahmen, wie beispielsweise Probleme mit der Netzwerkkonnektivität oder Ressourcenverfügbarkeit, die nach einer bestimmten Verzögerung automatisch wiederholt werden können. Mit einem KI-gesteuerten Orchestrator oder Ausnahmebehandler müssen Ihre Wiederholungsstrategien nicht mechanischer Natur sein und sich auf feste Algorithmen wie exponentiellen Rückzug verlassen. Sie können die Handhabung der Wiederholung dem “Ermessen” der KI-Komponente überlassen, die für die Entscheidung über den Umgang mit der Ausnahme verantwortlich ist.
3. **Fallback-Strategien:** Wenn eine KI-Komponente keine gültige Antwort liefern kann oder einen Fehler verursacht - ein häufiges Vorkommnis angesichts ihrer hochmodernen Natur - sollten Sie einen Fallback-Mechanismus einrichten, um sicherzustellen, dass der Workflow fortgesetzt werden kann. Dies könnte die Verwendung von Standardwerten, alternativen Algorithmen oder einem [Menschen im Regelkreis](#) umfassen, um Entscheidungen zu treffen und den Workflow voranzutreiben.
4. **Kompensationsmaßnahmen:** Die Anweisungen des Orchestrators sollten Instruktionen über Kompensationsmaßnahmen enthalten, um

Ausnahmen zu behandeln, die nicht automatisch gelöst werden können. Kompensationsmaßnahmen sind Schritte, die unternommen werden, um die Auswirkungen einer fehlgeschlagenen Operation rückgängig zu machen oder abzumildern. Wenn beispielsweise ein Zahlungsverarbeitungsschritt fehlschlägt, könnte eine Kompensationsmaßnahme darin bestehen, die Transaktion rückgängig zu machen und den Benutzer zu benachrichtigen. Kompensationsmaßnahmen helfen, die Datenkonsistenz und -integrität bei Ausnahmen aufrechtzuerhalten.

5. **Ausnahmeüberwachung und -alarmierung:** Richten Sie Überwachungs- und Alarmierungsmechanismen ein, um kritische Ausnahmen zu erkennen und relevante Interessengruppen zu benachrichtigen. Der Orchestrator kann über Schwellenwerte und Regeln informiert werden, um Alarme auszulösen, wenn Ausnahmen bestimmte Grenzen überschreiten oder wenn bestimmte Arten von Ausnahmen auftreten. Dies ermöglicht eine proaktive Identifizierung und Lösung von Problemen, bevor sie das Gesamtsystem beeinträchtigen.

Hier ist ein Beispiel für Ausnahmebehandlung und Wiederherstellung in einer Ruby-Workflow-Komponente:

```
1 class InventoryManager
2   def check_availability(order)
3     begin
4       # Perform inventory check logic
5       inventory = Inventory.find_by(product_id: order.product_id)
6       if inventory.available_quantity >= order.quantity
7         return true
8       else
9         raise InsufficientInventoryError,
10            "Insufficient inventory for product #{order.product_id}"
11      end
12    rescue InsufficientInventoryError => e
13      # Log the exception
14      logger.error("Inventory check failed: #{e.message}")
15
16      # Retry the operation after a delay
```

```
17     retry_count ||= 0
18     if retry_count < MAX_RETRIES
19         retry_count += 1
20         sleep(RETRY_DELAY)
21         retry
22     else
23         # Fallback to manual intervention
24         NotificationService.admin("Inventory check failed: Order #{order.id}")
25         return false
26     end
27 end
28 end
29 end
```

In diesem Beispiel überprüft die InventoryManager-Komponente die Verfügbarkeit eines Produkts für eine bestimmte Bestellung. Wenn die verfügbare Menge nicht ausreicht, wird eine InsufficientInventoryError ausgelöst. Die Ausnahme wird abgefangen, protokolliert und ein Wiederholungsmechanismus wird implementiert. Wenn das Wiederholungslimit überschritten wird, greift die Komponente auf manuelle Intervention zurück, indem sie einen Administrator benachrichtigt.

Durch die Implementierung robuster Ausnahmebehandlungs- und Wiederherstellungsmechanismen können Sie sicherstellen, dass Ihre intelligenten Workflows widerstandsfähig und wartbar sind sowie unerwartete Situationen elegant bewältigen können.

Diese Muster bilden die Grundlage der intelligenten Workflow-Orchestrierung und können kombiniert und an die spezifischen Anforderungen verschiedener Anwendungen angepasst werden. Durch die Nutzung dieser Muster können Entwickler Workflows erstellen, die flexibel und widerstandsfähig sind sowie für Leistung und Benutzererfahrung optimiert sind.

Im nächsten Abschnitt werden wir untersuchen, wie diese Muster in der Praxis implementiert werden können, wobei wir reale Beispiele und Code-Snippets

verwenden, um die Integration von KI-Komponenten in das Workflow-Management zu veranschaulichen.

Implementierung der intelligenten Workflow-Orchestrierung in der Praxis

Nachdem wir die wichtigsten Muster in der intelligenten Workflow-Orchestrierung erkundet haben, lassen Sie uns nun betrachten, wie diese Muster in realen Anwendungen implementiert werden können. Wir werden praktische Beispiele und Code-Snippets bereitstellen, um die Integration von KI-Komponenten in das Workflow-Management zu veranschaulichen.

Intelligenter Auftragsverarbeiter

Lassen Sie uns ein praktisches Beispiel für die Implementierung einer intelligenten Workflow-Orchestrierung anhand einer KI-gestützten `OrderProcessor`-Komponente in einer Ruby on Rails-E-Commerce-Anwendung betrachten. Der `OrderProcessor` verwirklicht das Konzept des [Process Manager Enterprise Integration](#), das wir erstmals in Kapitel 3 bei der Diskussion über [Multitude of Workers](#) kennengelernt haben. Die Komponente ist für die Verwaltung des Auftragsabwicklungs-Workflows verantwortlich, trifft Routing-Entscheidungen basierend auf Zwischenergebnissen und orchestriert die Ausführung verschiedener Verarbeitungsschritte.

Der Auftragsabwicklungsprozess umfasst mehrere Schritte wie Auftragsvalidierung, Bestandsprüfung, Zahlungsabwicklung und Versand. Jeder Schritt wird als separater Worker-Prozess implementiert, der eine bestimmte Aufgabe ausführt und das Ergebnis an den `OrderProcessor` zurückgibt. Die Schritte sind nicht obligatorisch und müssen nicht einmal unbedingt in einer bestimmten Reihenfolge ausgeführt werden.

Hier ist ein Beispiel für die Implementierung des `OrderProcessor`. Er enthält zwei Mixins von [Raix](#). Das erste (`ChatCompletion`) verleiht ihm die Fähigkeit

zur Chat-Vervollständigung, was ihn zu einer KI-Komponente macht. Das zweite (FunctionDispatch) ermöglicht den Funktionsaufruf durch die KI, sodass sie auf eine Aufforderung mit einem Funktionsaufruf statt einer Textnachricht reagieren kann.

Die Worker-Funktionen (validate_order, check_inventory, et al) delegieren an ihre jeweiligen Worker-Klassen, die KI- oder Nicht-KI-Komponenten sein können, wobei die einzige Anforderung darin besteht, dass sie die Ergebnisse ihrer Arbeit in einem Format zurückgeben, das als String dargestellt werden kann.



Wie bei allen anderen Beispielen in diesem Teil des Buches handelt es sich hierbei praktisch um Pseudo-Code, der nur dazu dient, die Bedeutung des Musters zu vermitteln und Ihre eigenen Kreationen zu inspirieren. Vollständige Beschreibungen der Muster und vollständige Code-Beispiele sind in Teil 2 enthalten.

```

1  class OrderProcessor
2      include Raix::ChatCompletion
3      include Raix::FunctionDispatch
4
5      SYSTEM_DIRECTIVE = "You are an order processor, tasked with..."
6
7      def initialize(order)
8          self.order = order
9          transcript << { system: SYSTEM_DIRECTIVE }
10         transcript << { user: order.to_json }
11     end
12
13     def perform
14         # will continue looping until `stop_looping!` is called
15         chat_completion(loop: true)
16     end
17
18     # list of functions available to be called by the AI
19     # truncated for brevity
20
21     def functions
22         [

```

```
23     {
24         name: "validate_order",
25         description: "Invoke to check validity of order",
26         parameters: {
27             ...
28         },
29         ...
30     ]
31 end
32
33 # implementation of functions that can be called by the AI
34 # entirely at its discretion, depending on the needs of the order
35
36 def validate_order
37     OrderValidationWorker.perform(@order)
38 end
39
40 def check_inventory
41     InventoryCheckWorker.perform(@order)
42 end
43
44 def process_payment
45     PaymentProcessingWorker.perform(@order)
46 end
47
48 def schedule_shipping
49     ShippingSchedulerWorker.perform(@order)
50 end
51
52 def send_confirmation
53     OrderConfirmationWorker.perform(@order)
54 end
55
56 def finished_processing
57     @order.update!(transcript:, processed_at: Time.current)
58     stop_looping!
59 end
60 end
```

In diesem Beispiel wird der OrderProcessor mit einem Bestellobjekt initialisiert und führt ein Protokoll der Workflow-Ausführung im typischen Konversationsformat,

das für große Sprachmodelle charakteristisch ist. Die KI erhält die vollständige Kontrolle über die Orchestrierung der verschiedenen Verarbeitungsschritte, wie Bestellungsvalidierung, Bestandsprüfung, Zahlungsabwicklung und Versand.

Jedes Mal, wenn die Methode `chat_completion` aufgerufen wird, wird das Protokoll an die KI gesendet, damit diese eine Vervollständigung in Form eines Funktionsaufrufs bereitstellt. Es liegt vollständig in der Verantwortung der KI, das Ergebnis des vorherigen Schritts zu analysieren und die entsprechende Aktion zu bestimmen. Wenn beispielsweise die Bestandsprüfung einen niedrigen Lagerbestand aufzeigt, kann der `OrderProcessor` eine Nachbestellungsaufgabe einplanen. Falls die Zahlungsabwicklung fehlschlägt, kann er einen erneuten Versuch starten oder den Kundenservice benachrichtigen.

Das obige Beispiel enthält zwar keine definierten Funktionen für Nachbestellung oder Kundenservice-Benachrichtigung, könnte diese aber durchaus haben.

Das Protokoll wächst mit jedem Funktionsaufruf und dient als Aufzeichnung der Workflow-Ausführung, einschließlich der Ergebnisse jedes Schritts und der KI-generierten Anweisungen für die nächsten Schritte. Dieses Protokoll kann für Fehlerbehebung, Prüfung und Einblick in den Bestellabwicklungsprozess verwendet werden.

Durch den Einsatz von KI im `OrderProcessor` kann die E-Commerce-Anwendung den Workflow dynamisch an Echtzeitdaten anpassen und Ausnahmen intelligent behandeln. Die KI-Komponente kann fundierte Entscheidungen treffen, den Workflow optimieren und eine reibungslose Bestellverarbeitung auch in komplexen Szenarien gewährleisten.

Da die einzige Anforderung an die Arbeitsprozesse darin besteht, eine verständliche Ausgabe zu liefern, die die KI bei der Entscheidung über das weitere Vorgehen berücksichtigen kann, wird möglicherweise deutlich, wie dieser Ansatz den Aufwand

für die Eingabe-/Ausgabe-Zuordnung reduzieren kann, der typischerweise bei der Integration verschiedener Systeme anfällt.

Intelligenter Inhaltsmoderator

Social-Media-Anwendungen erfordern im Allgemeinen mindestens eine grundlegende Inhaltsmoderation, um eine sichere und gesunde Community zu gewährleisten. Dieses Beispiel einer ContentModerator-Komponente nutzt KI, um den Moderationsworkflow intelligent zu orchestrieren und trifft Entscheidungen basierend auf den Eigenschaften der Inhalte und den Ergebnissen verschiedener Moderationsschritte.

Der Moderationsprozess umfasst mehrere Schritte wie Textanalyse, Bilderkennung, Bewertung der Benutzerreputation und manuelle Überprüfung. Jeder Schritt wird als separater Arbeitsprozess implementiert, der eine spezifische Aufgabe ausführt und das Ergebnis an den ContentModerator zurückgibt.

Hier ist eine Beispielimplementierung des ContentModerator:

```
1 class ContentModerator
2   include Raix::ChatCompletion
3   include Raix::FunctionDispatch
4
5   SYSTEM_DIRECTIVE = "You are a content moderator process manager,
6     tasked with the workflow involved in moderating user-generated content..."
7
8   def initialize(content)
9     @content = content
10    @transcript = [
11      { system: SYSTEM_DIRECTIVE },
12      { user: content.to_json }
13    ]
14  end
15
16  def perform
17    complete(@transcript)
18  end
```

```
19
20 def model
21     "openai/gpt-4"
22 end
23
24 # list of functions available to be called by the AI
25 # truncated for brevity
26
27 def functions
28     [
29         {
30             name: "analyze_text",
31             # ...
32         },
33         {
34             name: "recognize_image",
35             description: "Invoke to describe images...",
36             # ...
37         },
38         {
39             name: "assess_user_reputation",
40             # ...
41         },
42         {
43             name: "escalate_to_manual_review",
44             # ...
45         },
46         {
47             name: "approve_content",
48             # ...
49         },
50         {
51             name: "reject_content",
52             # ...
53         }
54     ]
55 end
56
57 # implementation of functions that can be called by the AI
58 # entirely at its discretion, depending on the needs of the order
59
60 def analyze_text
```



```
61     result = TextAnalysisWorker.perform(@content)
62     continue_with(result)
63 end
64
65 def recognize_image
66     result = ImageRecognitionWorker.perform(@content)
67     continue_with(result)
68 end
69
70 def assess_user_reputation
71     result = UserReputationWorker.perform(@content.user)
72     continue_with(result)
73 end
74
75 def escalate_to_manual_review
76     ManualReviewWorker.perform(@content)
77     @content.update!(status: 'pending', transcript: @transcript)
78 end
79
80 def approve_content
81     @content.update!(status: 'approved', transcript: @transcript)
82 end
83
84 def reject_content
85     @content.update!(status: 'rejected', transcript: @transcript)
86 end
87
88 private
89
90 def continue_with(result)
91     @transcript << { function: result }
92     complete(@transcript)
93 end
94 end
```

In diesem Beispiel wird der ContentModerator mit einem Inhaltsobjekt initialisiert und führt ein Moderationsprotokoll im Gesprächsformat. Die KI-Komponente hat die vollständige Kontrolle über den Moderationsarbeitsablauf und entscheidet basierend auf den Eigenschaften des Inhalts und den Ergebnissen jedes Schritts, welche Schritte ausgeführt werden sollen.

Die verfügbaren Arbeitsfunktionen, die die KI aufrufen kann, umfassen `analyze_text`, `recognize_image`, `assess_user_reputation` und `escalate_to_manual_review`. Jede Funktion delegiert die Aufgabe an einen entsprechenden Arbeitsprozess (`TextAnalysisWorker`, `ImageRecognitionWorker`, usw.) und fügt das Ergebnis dem Moderationsprotokoll hinzu, mit Ausnahme der Eskalationsfunktion, die als Endzustand fungiert. Auch die Funktionen `approve_content` und `reject_content` fungieren als Endzustände.

Die KI-Komponente analysiert den Inhalt und bestimmt die angemessene Vorgehensweise. Wenn der Inhalt Bildreferenzen enthält, kann sie den `recognize_image`-Worker zur Unterstützung bei einer visuellen Überprüfung aufrufen. Wenn ein Worker vor potenziell schädlichen Inhalten warnt, kann die KI entscheiden, den Inhalt zur manuellen Überprüfung weiterzuleiten oder ihn direkt abzulehnen. Je nach Schwere der Warnung kann die KI aber auch die Ergebnisse der Benutzerreputationsbewertung bei der Entscheidung berücksichtigen, wie sie mit Inhalten umgeht, bei denen sie sich nicht sicher ist. Je nach Anwendungsfall haben vertrauenswürdige Benutzer möglicherweise mehr Spielraum bei dem, was sie posten können. Und so weiter und so fort...

Wie beim vorherigen Beispiel des Prozessmanagers dient das Moderationsprotokoll als Aufzeichnung der Workflow-Ausführung, einschließlich der Ergebnisse jedes Schritts und der KI-generierten Entscheidungen. Dieses Protokoll kann für Audits, Transparenz und die Verbesserung des Moderationsprozesses im Laufe der Zeit verwendet werden.

Durch den Einsatz von KI im ContentModerator kann die Social-Media-Anwendung den Moderationsarbeitsablauf dynamisch an die Eigenschaften der Inhalte anpassen und komplexe Moderationsszenarien intelligent handhaben. Die KI-Komponente kann fundierte Entscheidungen treffen, den Arbeitsablauf optimieren und eine sichere und gesunde Community-Erfahrung gewährleisten.

Lassen Sie uns zwei weitere Beispiele betrachten, die die prädiktive Aufgabenplanung sowie die Ausnahmebehandlung und -wiederherstellung im Kontext der intelligenten

Workflow-Orchestrierung demonstrieren.

Prädiktive Aufgabenplanung in einem Kundenbetreuungssystem

In einer mit Ruby on Rails entwickelten Kundenbetreuungsanwendung ist die effiziente Verwaltung und Priorisierung von Support-Tickets entscheidend für eine zeitnahe Unterstützung der Kunden. Die `SupportTicketScheduler`-Komponente nutzt KI, um Support-Tickets vorausschauend zu planen und verfügbaren Mitarbeitern zuzuweisen, basierend auf verschiedenen Faktoren wie Dringlichkeit des Tickets, Expertise der Mitarbeiter und Arbeitsauslastung.

```
1  class SupportTicketScheduler
2    include Raix::ChatCompletion
3    include Raix::FunctionDispatch
4
5    SYSTEM_DIRECTIVE = "You are a support ticket scheduler,
6      tasked with intelligently assigning tickets to available agents..."
7
8    def initialize(ticket)
9      @ticket = ticket
10     @transcript = [
11       { system: SYSTEM_DIRECTIVE },
12       { user: ticket.to_json }
13     ]
14   end
15
16   def perform
17     complete(@transcript)
18   end
19
20   def model
21     "openai/gpt-4"
22   end
23
24   def functions
25     [
```

```
26     {
27         name: "analyze_ticket_urgency",
28         # ...
29     },
30     {
31         name: "list_available_agents",
32         description: "Includes expertise of available agents",
33         # ...
34     },
35     {
36         name: "predict_agent_workload",
37         description: "Uses historical data to predict upcoming workloads",
38         # ...
39     },
40     {
41         name: "assign_ticket_to_agent",
42         # ...
43     },
44     {
45         name: "reschedule_ticket",
46         # ...
47     }
48 ]
49 end
50
51 # implementation of functions that can be called by the AI
52 # entirely at its discretion, depending on the needs of the order
53
54 def analyze_ticket_urgency
55     result = TicketUrgencyAnalyzer.perform(@ticket)
56     continue_with(result)
57 end
58
59 def list_available_agents
60     result = ListAvailableAgents.perform
61     continue_with(result)
62 end
63
64 def predict_agent_workload
65     result = AgentWorkloadPredictor.perform
66     continue_with(result)
67 end
```

```
68
69  def assign_ticket_to_agent
70      TicketAssigner.perform(@ticket, @transcript)
71  end
72
73  def delay_assignment(until)
74      until = DateTimeStandardizer.process(until)
75      SupportTicketScheduler.delay(@ticket, @transcript, until)
76  end
77
78  private
79
80  def continue_with(result)
81      @transcript << { function: result }
82      complete(@transcript)
83  end
84  end
```

In diesem Beispiel wird der SupportTicketScheduler mit einem Support-Ticket-Objekt initialisiert und führt ein Planungsprotokoll. Die KI-Komponente analysiert die Ticket-Details und plant die Ticket-Zuweisung prädiktiv auf Basis von Faktoren wie Ticket-Dringlichkeit, Mitarbeiter-Expertise und prognostizierter Mitarbeiterauslastung.

Die für die KI verfügbaren Funktionen umfassen `analyze_ticket_urgency`, `list_available_agents`, `predict_agent_workload` und `assign_ticket_to_agent`. Jede Funktion delegiert die Aufgabe an eine entsprechende Analyse- oder Prognosekomponente und fügt das Ergebnis dem Planungsprotokoll hinzu. Die KI hat auch die Möglichkeit, die Zuweisung mittels der Funktion `delay_assignment` zu verzögern.

Die KI-Komponente untersucht das Planungsprotokoll und trifft fundierte Entscheidungen zur Ticket-Zuweisung. Sie berücksichtigt die Dringlichkeit des Tickets, die Expertise der verfügbaren Mitarbeiter und die prognostizierte Arbeitsbelastung jedes Mitarbeiters, um den am besten geeigneten Mitarbeiter für die Bearbeitung des Tickets zu bestimmen.

Durch den Einsatz der prädiktiven Aufgabenplanung kann die Kundenservice-

Anwendung die Ticket-Zuweisung optimieren, Reaktionszeiten verkürzen und die allgemeine Kundenzufriedenheit verbessern. Die proaktive und effiziente Verwaltung von Support-Tickets stellt sicher, dass die richtigen Tickets zur richtigen Zeit an die richtigen Mitarbeiter zugewiesen werden.

Ausnahmebehandlung und Wiederherstellung in einer Datenverarbeitungs-Pipeline

Die Behandlung von Ausnahmen und die Wiederherstellung nach Fehlern sind essentiell, um die Datenintegrität zu gewährleisten und Datenverlust zu verhindern. Die `DataProcessingOrchestrator`-Komponente nutzt KI, um intelligent mit Ausnahmen umzugehen und den Wiederherstellungsprozess in einer Datenverarbeitungs-Pipeline zu orchestrieren

```
1 class DataProcessingOrchestrator
2     include Raix::ChatCompletion
3     include Raix::FunctionDispatch
4
5     SYSTEM_DIRECTIVE = "You are a data processing orchestrator..."
6
7     def initialize(data_batch)
8         @data_batch = data_batch
9         @transcript = [
10             { system: SYSTEM_DIRECTIVE },
11             { user: data_batch.to_json }
12         ]
13     end
14
15     def perform
16         complete(@transcript)
17     end
18
19     def model
20         "openai/gpt-4"
21     end
22
23     def functions
```

```
24     [
25         {
26             name: "validate_data",
27             # ...
28         },
29         {
30             name: "process_data",
31             # ...
32         },
33         {
34             name: "request_fix",
35             # ...
36         },
37         {
38             name: "retry_processing",
39             # ...
40         },
41         {
42             name: "mark_data_as_failed",
43             # ...
44         },
45         {
46             name: "finished",
47             # ...
48         }
49     ]
50 end
51
52 # implementation of functions that can be called by the AI
53 # entirely at its discretion, depending on the needs of the order
54
55 def validate_data
56     result = DataValidator.perform(@data_batch)
57     continue_with(result)
58 rescue ValidationException => e
59     handle_validation_exception(e)
60 end
61
62 def process_data
63     result = DataProcessor.perform(@data_batch)
64     continue_with(result)
65 rescue ProcessingException => e
```

```
66     handle_processing_exception(e)
67 end
68
69 def request_fix(description_of_fix)
70     result = SmartDataFixer.new(description_of_fix, @data_batch)
71     continue_with(result)
72 end
73
74 def retry_processing(timeout_in_seconds)
75     wait(timeout_in_seconds)
76     process_data
77 end
78
79 def mark_data_as_failed
80     @data_batch.update!(status: 'failed', transcript: @transcript)
81 end
82
83 def finished
84     @data_batch.update!(status: 'finished', transcript: @transcript)
85 end
86
87 private
88
89 def continue_with(result)
90     @transcript << { function: result }
91     complete(@transcript)
92 end
93
94 def handle_validation_exception(exception)
95     @transcript << { exception: exception.message }
96     complete(@transcript)
97 end
98
99 def handle_processing_exception(exception)
100     @transcript << { exception: exception.message }
101     complete(@transcript)
102 end
103 end
```

In diesem Beispiel wird der `DataProcessingOrchestrator` mit einem Daten-Batch-Objekt initialisiert und führt ein Verarbeitungsprotokoll. Die KI-Komponente

orchestriert die Datenverarbeitungspipeline, behandelt Ausnahmen und erholt sich nach Bedarf von Fehlern.

Die der KI zur Verfügung stehenden Funktionen umfassen `validate_data`, `process_data`, `request_fix`, `retry_processing` und `mark_data_as_failed`. Jede Funktion delegiert die Aufgabe an eine entsprechende Datenverarbeitungskomponente und fügt das Ergebnis oder die Ausnahmedetails dem Verarbeitungsprotokoll hinzu.

Wenn während des `validate_data`-Schritts eine Validierungsausnahme auftritt, fügt die `handle_validation_exception`-Funktion die Ausnahmedaten dem Protokoll hinzu und gibt die Kontrolle an die KI zurück. Ähnlich verhält es sich, wenn während des `process_data`-Schritts eine Verarbeitungsausnahme auftritt - hier kann die KI über die Wiederherstellungsstrategie entscheiden.

Je nach Art der aufgetretenen Ausnahme kann die KI nach eigenem Ermessen entscheiden, `request_fix` aufzurufen, was an eine KI-gesteuerte `SmartDataFixer`-Komponente delegiert wird (siehe Kapitel über Selbstheilende Daten). Der Datenkorrektor erhält eine einfache Beschreibung in natürlicher Sprache, wie er den `@data_batch` modifizieren soll, damit die Verarbeitung wiederholt werden kann. Vielleicht würde eine erfolgreiche Wiederholung bedeuten, dass Datensätze aus dem Daten-Batch entfernt werden, die die Validierung nicht bestanden haben, und/oder dass sie zur manuellen Überprüfung in eine andere Verarbeitungspipeline kopiert werden? Die Möglichkeiten sind nahezu endlos.

Durch die Integration von KI-gesteuerter Ausnahmebehandlung und Wiederherstellung wird die Datenverarbeitungsanwendung robuster und fehlertoleranter. Der `DataProcessingOrchestrator` verwaltet Ausnahmen intelligent, minimiert Datenverluste und gewährleistet die reibungslose Ausführung des Datenverarbeitungsworkflows.

Überwachung und Protokollierung

Überwachung und Protokollierung bieten Einblick in den Fortschritt, die Leistung und den Zustand von KI-gesteuerten Workflow-Komponenten und ermöglichen es Entwicklern, das Verhalten des Systems zu verfolgen und zu analysieren. Die Implementierung effektiver Überwachungs- und Protokollierungsmechanismen ist essentiell für das Debugging, die Prüfung und die kontinuierliche Verbesserung intelligenter Workflows.

Überwachung des Workflow-Fortschritts und der Leistung

Um die reibungslose Ausführung intelligenter Workflows sicherzustellen, ist es wichtig, den Fortschritt und die Leistung jeder Workflow-Komponente zu überwachen. Dies beinhaltet die Verfolgung wichtiger Metriken und Ereignisse während des gesamten Workflow-Lebenszyklus.

Wichtige zu überwachende Aspekte sind:

- 1. Workflow-Ausführungszeit:** Messen der Zeit, die jede Workflow-Komponente für die Ausführung ihrer Aufgabe benötigt. Dies hilft dabei, Leistungsengpässe zu identifizieren und die Gesamteffizienz des Workflows zu optimieren.
- 2. Ressourcennutzung:** Überwachung der Nutzung von Systemressourcen wie CPU, Arbeitsspeicher und Speicherplatz durch jede Workflow-Komponente. Dies hilft sicherzustellen, dass das System innerhalb seiner Kapazität arbeitet und die Arbeitslast effektiv bewältigen kann.
- 3. Fehlerraten und Ausnahmen:** Verfolgung des Auftretens von Fehlern und Ausnahmen innerhalb der Workflow-Komponenten. Dies hilft dabei, potenzielle Probleme zu identifizieren und ermöglicht proaktive Fehlerbehandlung und Wiederherstellung.

4. Entscheidungspunkte und Ergebnisse: Überwachung der Entscheidungspunkte innerhalb des Workflows und der Ergebnisse KI-gesteuerter Entscheidungen. Dies liefert Einblicke in das Verhalten und die Effektivität der KI-Komponenten.

Die durch Überwachungsprozesse erfassten Daten können in Dashboards angezeigt oder als Eingaben für geplante Berichte verwendet werden, die Systemadministratoren über den Systemzustand informieren.



Überwachungsdaten können an einen KI-gesteuerten Systemadministrationsprozess zur Überprüfung und möglichen Maßnahmen weitergeleitet werden!

Protokollierung wichtiger Ereignisse und Entscheidungen

Die Protokollierung ist eine wesentliche Praxis, bei der relevante Informationen über wichtige Ereignisse, Entscheidungen und Ausnahmen während der Workflow-Ausführung erfasst und gespeichert werden.

Wichtige zu protokollierende Aspekte sind:

1. Workflow-Initiierung und -Abschluss: Protokollierung der Start- und Endzeiten jeder Workflow-Instanz zusammen mit relevanten Metadaten wie den Eingabedaten und dem Benutzerkontext.

2. Komponentenausführung: Protokollierung der Ausführungsdetails jeder Workflow-Komponente, einschließlich der Eingabeparameter, Ausgabeergebnisse und aller erzeugten Zwischendaten.

3. KI-Entscheidungen und -Begründungen: Protokollierung der von KI-Komponenten getroffenen Entscheidungen zusammen mit den zugrundeliegenden Begründungen oder Konfidenzwerten. Dies sorgt für Transparenz und ermöglicht die Überprüfung KI-gesteuerter Entscheidungen.

4. Ausnahmen und Fehlermeldungen: Protokollierung aller während der Workflow-Ausführung aufgetretenen Ausnahmen oder Fehlermeldungen, einschließlich des Stack-Trace und relevanter Kontextinformationen.

Die Protokollierung kann mit verschiedenen Techniken implementiert werden, wie dem Schreiben in Protokolldateien, dem Speichern von Protokollen in einer Datenbank oder dem Senden von Protokollen an einen zentralisierten Protokollierungsdienst. Es ist wichtig, ein Protokollierungsframework zu wählen, das Flexibilität, Skalierbarkeit und einfache Integration in die Anwendungsarchitektur bietet.

Hier ist ein Beispiel dafür, wie die Protokollierung in einer Ruby on Rails-Anwendung mit der Klasse `ActiveSupport::Logger` implementiert werden kann:

```
1 class WorkflowLogger
2   def self.log(message, severity = :info)
3     @logger ||= ActiveSupport::Logger.new('workflow.log')
4     @logger.formatter ||= proc do |severity, datetime, progname, msg|
5       "#{datetime} [{severity}] #{msg}\n"
6     end
7     @logger.send(severity, message)
8   end
9 end
10
11 # Usage example
12 WorkflowLogger.log("Workflow initiated for order #{@order.id}")
13 WorkflowLogger.log("Payment processing completed successfully")
14 WorkflowLogger.log("Inventory check failed for item #{item.id}", :error)
```

Durch die strategische Platzierung von Protokollierungsanweisungen in den Workflow-Komponenten und KI-Entscheidungspunkten können Entwickler wertvolle Informationen für Fehlersuche, Prüfung und Analyse erfassen.

Vorteile von Überwachung und Protokollierung

Die Implementierung von Überwachung und Protokollierung in der intelligenten Workflow-Orchestrierung bietet mehrere Vorteile:

1. Fehlersuche und Fehlerbehebung: Detaillierte Protokolle und Überwachungsdaten helfen Entwicklern dabei, Probleme schnell zu identifizieren und zu diagnostizieren. Sie bieten Einblicke in den Workflow-Ausführungsablauf, Komponenteninteraktionen und aufgetretene Fehler oder Ausnahmen.

2. Leistungsoptimierung: Die Überwachung von Leistungsmetriken ermöglicht es Entwicklern, Engpässe zu identifizieren und die Workflow-Komponenten für bessere Effizienz zu optimieren. Durch die Analyse von Ausführungszeiten, Ressourcennutzung und anderen Metriken können Entwickler fundierte Entscheidungen zur Verbesserung der Gesamtleistung des Systems treffen.

3. Prüfung und Compliance: Die Protokollierung wichtiger Ereignisse und Entscheidungen bietet einen Prüfpfad für regulatorische Compliance und Verantwortlichkeit. Dies ermöglicht es Organisationen, die von KI-Komponenten getroffenen Maßnahmen zu verfolgen und zu überprüfen sowie die Einhaltung von Geschäftsregeln und rechtlichen Anforderungen sicherzustellen.

4. Kontinuierliche Verbesserung: Überwachungs- und Protokollierungsdaten dienen als wertvolle Eingaben für die kontinuierliche Verbesserung intelligenter Workflows. Durch die Analyse historischer Daten, die Identifizierung von Mustern und die Messung der Effektivität von KI-Entscheidungen können Entwickler die Workflow-Orchestrierungslogik iterativ verfeinern und verbessern.

Überlegungen und Best Practices

Bei der Implementierung von Überwachung und Protokollierung in der intelligenten Workflow-Orchestrierung sind folgende Best Practices zu beachten:

1. Klare Überwachungsmetriken definieren: Identifizieren Sie die wichtigsten Metriken und Ereignisse, die basierend auf den spezifischen Anforderungen des Workflows überwacht werden müssen. Konzentrieren Sie sich auf Metriken, die aussagekräftige Einblicke in die Leistung, den Zustand und das Verhalten des Systems liefern.

2. Granulare Protokollierung implementieren: Stellen Sie sicher, dass Protokollierungsanweisungen an geeigneten Stellen innerhalb der Workflow-Komponenten und KI-Entscheidungspunkte platziert sind. Erfassen Sie relevante Kontextinformationen wie Eingabeparameter, Ausgabeergebnisse und generierte Zwischendaten.

3. Strukturierte Protokollierung verwenden: Verwenden Sie ein strukturiertes Protokollierungsformat, um das Parsen und die Analyse von Protokolldaten zu erleichtern. Strukturierte Protokollierung ermöglicht eine bessere Durchsuchbarkeit, Filterung und Aggregation von Protokolleinträgen.

4. Protokollaufbewahrung und -rotation verwalten: Implementieren Sie Richtlinien für die Protokollaufbewahrung und -rotation, um die Speicherung und den Lebenszyklus von Protokolldateien zu verwalten. Legen Sie die angemessene Aufbewahrungsfrist basierend auf gesetzlichen Anforderungen, Speicherbeschränkungen und Analysebedarf fest. Wenn möglich, lagern Sie die Protokollierung an einen Drittanbieterdienst wie [Papertrail](#) aus.

5. Sensible Informationen schützen: Seien Sie vorsichtig bei der Protokollierung sensibler Informationen wie personenbezogener Daten (PII) oder vertraulicher Geschäftsdaten. Implementieren Sie geeignete Sicherheitsmaßnahmen wie Datenmasking oder Verschlüsselung zum Schutz sensibler Informationen in Protokolldateien.

6. Integration mit Überwachungs- und Alarmierungswerkzeugen: Nutzen Sie Überwachungs- und Alarmierungswerkzeuge zur Zentralisierung der Sammlung, Analyse und Visualisierung von Überwachungs- und Protokollierungsdaten. Diese Tools können Echtzeit-Einblicke liefern, Warnungen basierend auf vordefinierten Schwellenwerten generieren und die proaktive Erkennung und Behebung von Problemen erleichtern. Mein Lieblingswerkzeug dafür ist [Datadog](#).

Durch die Implementierung umfassender Überwachungs- und Protokollierungsmechanismen können Entwickler wertvolle Einblicke in das Verhalten und die Leistung intelligenter

Workflows gewinnen. Diese Einblicke ermöglichen eine effektive Fehlersuche, Optimierung und kontinuierliche Verbesserung von KI-gestützten Workflow-Orchestrierungssystemen.

Skalierbarkeits- und Leistungsüberlegungen

Skalierbarkeit und Leistung sind kritische Aspekte, die bei der Konzeption und Implementierung intelligenter Workflow-Orchestrierungssysteme zu berücksichtigen sind. Mit zunehmendem Volumen gleichzeitiger Workflows und steigender Komplexität KI-gestützter Komponenten wird es essentiell, sicherzustellen, dass das System die Arbeitslast effizient bewältigen und sich nahtlos an wachsende Anforderungen anpassen kann.

Umgang mit hohen Volumen gleichzeitiger Workflows

Intelligente Workflow-Orchestrierungssysteme müssen oft eine große Anzahl gleichzeitiger Workflows bewältigen. Um Skalierbarkeit zu gewährleisten, sollten folgende Strategien berücksichtigt werden:

1. Asynchrone Verarbeitung: Implementieren Sie asynchrone Verarbeitungsmechanismen, um die Ausführung von Workflow-Komponenten zu entkoppeln. Dies ermöglicht es dem System, mehrere Workflows gleichzeitig zu bearbeiten, ohne dass einzelne Komponenten blockieren oder auf deren Abschluss gewartet werden muss. Asynchrone Verarbeitung kann durch Nachrichtenwarteschlangen, ereignisgesteuerte Architekturen oder Hintergrundverarbeitungs-Frameworks wie Sidekiq erreicht werden.

2. Verteilte Architektur: Gestalten Sie die Systemarchitektur so, dass sie serverlose Komponenten (wie AWS Lambda) verwendet oder die Arbeitslast einfach auf mehrere Knoten oder Server neben Ihrem Hauptanwendungsserver verteilt. Dies ermöglicht horizontale Skalierbarkeit, bei der zusätzliche Knoten hinzugefügt werden können, um erhöhte Workflow-Volumen zu bewältigen.

3. Parallele Ausführung: Identifizieren Sie Möglichkeiten zur parallelen Ausführung innerhalb von Workflows. Einige Workflow-Komponenten können unabhängig voneinander sein und gleichzeitig ausgeführt werden. Durch die Nutzung paralleler Verarbeitungstechniken wie Multi-Threading oder verteilte Task-Warteschlangen kann das System die Ressourcennutzung optimieren und die gesamte Workflow-Ausführungszeit reduzieren.

Optimierung der Leistung von KI-gestützten Komponenten

KI-gestützte Komponenten wie maschinelle Lernmodelle oder Systeme zur Verarbeitung natürlicher Sprache können rechenintensiv sein und die Gesamtleistung des Workflow-Orchestrierungssystems beeinträchtigen. Um die Leistung von KI-Komponenten zu optimieren, sollten folgende Techniken in Betracht gezogen werden:

- 1. Caching:** Wenn Ihre KI-Verarbeitung rein generativ ist und keine Echtzeit-Informationsabfragen oder externe Integrationen für die Erstellung von Chat-Antworten erfordert, können Sie Caching-Mechanismen zur Speicherung und Wiederverwendung der Ergebnisse häufig genutzter oder rechenintensiver Operationen in Betracht ziehen.
- 2. Modelloptimierung:** Optimieren Sie kontinuierlich die Verwendung der KI-Modelle in Workflow-Komponenten. Dies kann Techniken wie *Prompt-Distillation* umfassen oder einfach das Testen neuer Modelle, sobald diese verfügbar werden.
- 3. Batch-Verarbeitung:** Wenn Sie mit GPT-4-Klasse-Modellen arbeiten, können Sie möglicherweise Batch-Verarbeitungstechniken nutzen, um mehrere Datenpunkte oder Anfragen in einem einzigen Durchgang zu verarbeiten, anstatt sie einzeln zu bearbeiten. Durch die Verarbeitung von Daten in Batches kann das System die Ressourcennutzung optimieren und den Overhead wiederholter Modellanfragen reduzieren.

Überwachung und Profiling der Leistung

Um Leistungsengpässe zu identifizieren und die Skalierbarkeit des intelligenten Workflow-Orchestrierungssystems zu optimieren, ist es entscheidend, Überwachungs- und Profiling-Mechanismen zu implementieren. Berücksichtigen Sie folgende Ansätze:

1. Leistungskennzahlen: Definieren und verfolgen Sie wichtige Leistungskennzahlen wie Antwortzeit, Durchsatz, Ressourcennutzung und Latenz. Diese Metriken liefern Einblicke in die Systemleistung und helfen bei der Identifizierung von Optimierungsbereichen. Der beliebte KI-Modell-Aggregator [OpenRouter](#) enthält Host¹- und Speed²-Metriken in jeder API-Antwort, wodurch die Verfolgung dieser wichtigen Kennzahlen trivial wird.

2. Profiling-Tools: Nutzen Sie Profiling-Tools zur Analyse der Leistung einzelner Workflow-Komponenten und KI-Operationen. Profiling-Tools können helfen, Leistungs-Hotspots, ineffiziente Code-Pfade oder ressourcenintensive Operationen zu identifizieren. Beliebte Profiling-Tools sind New Relic, Scout oder die in der Programmiersprache oder dem Framework integrierten Profiler.

3. Lasttests: Führen Sie Lasttests durch, um die Systemleistung unter verschiedenen gleichzeitigen Arbeitslasten zu bewerten. Lasttests helfen dabei, die Skalierungsgrenzen des Systems zu identifizieren, Leistungseinbußen zu erkennen und sicherzustellen, dass das System den erwarteten Datenverkehr ohne Leistungseinbußen bewältigen kann.

4. Kontinuierliche Überwachung: Implementieren Sie kontinuierliche Überwachungs- und Alarmmechanismen, um Leistungsprobleme und Engpässe proaktiv zu erkennen. Richten Sie Überwachungs-Dashboards und Alarmer ein, um wichtige Leistungsindikatoren (KPIs) zu verfolgen und Benachrichtigungen zu erhalten, wenn vordefinierte Schwellenwerte überschritten werden. Dies ermöglicht eine schnelle Identifizierung und Behebung von Leistungsproblemen.

¹Host ist die Zeit, die benötigt wurde, um das erste Byte der gestreamten Generierung vom Modell-Host zu empfangen, auch bekannt als "Time to First Byte".

²Speed wird berechnet als die Anzahl der Completion-Tokens geteilt durch die gesamte Generierungszeit. Bei nicht-gestreamten Anfragen wird die Latenz als Teil der Generierungszeit betrachtet.

Skalierungsstrategien

Um steigende Arbeitslasten zu bewältigen und die Skalierbarkeit des intelligenten Workflow-Orchestrierungssystems sicherzustellen, sollten folgende Skalierungsstrategien in Betracht gezogen werden:

1. Vertikale Skalierung: Vertikale Skalierung beinhaltet die Erhöhung der Ressourcen (z.B. CPU, Speicher) einzelner Knoten oder Server, um höhere Arbeitslasten zu bewältigen. Dieser Ansatz ist geeignet, wenn das System mehr Rechenleistung oder Speicher benötigt, um komplexe Workflows oder KI-Operationen zu verarbeiten.

2. Horizontale Skalierung: Horizontale Skalierung beinhaltet das Hinzufügen weiterer Knoten oder Server zum System, um die Arbeitslast zu verteilen. Dieser Ansatz ist effektiv, wenn das System eine große Anzahl gleichzeitiger Workflows verarbeiten muss oder wenn die Arbeitslast leicht auf mehrere Knoten verteilt werden kann. Horizontale Skalierung erfordert eine verteilte Architektur und Lastausgleichsmechanismen, um eine gleichmäßige Verkehrsverteilung sicherzustellen.

3. Automatische Skalierung: Implementieren Sie automatische Skalierungsmechanismen, um die Anzahl der Knoten oder Ressourcen basierend auf der Arbeitsauslastung automatisch anzupassen. Automatische Skalierung ermöglicht es dem System, sich je nach eingehendem Datenverkehr dynamisch nach oben oder unten zu skalieren und dabei eine optimale Ressourcennutzung und Kosteneffizienz sicherzustellen. Cloud-Plattformen wie Amazon Web Services (AWS) oder Google Cloud Platform (GCP) bieten automatische Skalierungsfunktionen, die für intelligente Workflow-Orchestrierungssysteme genutzt werden können.

Leistungsoptimierungstechniken

Zusätzlich zu den Skalierungsstrategien sollten folgende Leistungsoptimierungstechniken in Betracht gezogen werden, um die Effizienz des intelligenten Workflow-Orchestrierungssystems zu verbessern:

1. Effiziente Datenspeicherung und -abruf: Optimieren Sie die von den Workflow-Komponenten verwendeten Mechanismen zur Datenspeicherung und zum Datenabruf. Verwenden Sie effiziente Datenbankindizierung, Abfrageoptimierungstechniken und Daten-Caching, um die Latenz zu minimieren und die Leistung datenintensiver Operationen zu verbessern.

2. Asynchrone E/A: Nutzen Sie asynchrone Ein-/Ausgabe-Operationen, um Blockierungen zu vermeiden und die Reaktionsfähigkeit des Systems zu verbessern. Asynchrone E/A ermöglicht es dem System, mehrere Anfragen gleichzeitig zu verarbeiten, ohne auf den Abschluss von E/A-Operationen warten zu müssen, wodurch die Ressourcennutzung maximiert wird.

3. Effiziente Serialisierung und Deserialisierung: Optimieren Sie die Serialisierungs- und Deserialisierungsprozesse, die für den Datenaustausch zwischen Workflow-Komponenten verwendet werden. Verwenden Sie effiziente Serialisierungsformate wie Protocol Buffers oder MessagePack, um den Overhead der Datenserialisierung zu reduzieren und die Leistung der Kommunikation zwischen den Komponenten zu verbessern.



Für Ruby-basierte Anwendungen sollten Sie die Verwendung von [Universal ID](#) in Betracht ziehen. Universal ID nutzt sowohl MessagePack als auch Brotli (eine Kombination, die für Geschwindigkeit und erstklassige Datenkompression entwickelt wurde). In Kombination sind diese Bibliotheken bis zu 30% schneller und erreichen Kompressionsraten, die nur 2-5% von Protocol Buffers abweichen.

4. Komprimierung und Kodierung: Wenden Sie Kompressions- und Kodierungstechniken an, um die Größe der zwischen Workflow-Komponenten übertragenen Daten zu reduzieren. Kompressionsalgorithmen wie gzip oder Brotli können den Netzwerkbandbreitenverbrauch erheblich reduzieren und die Gesamtleistung des Systems verbessern.

Durch die Berücksichtigung von Skalierbarkeits- und Leistungsaspekten während der Konzeption und Implementierung intelligenter Workflow-Orchestrierungssysteme können Sie sicherstellen, dass Ihr System hohe Volumina gleichzeitiger Workflows bewältigen, die Leistung KI-gestützter Komponenten optimieren und nahtlos skalieren kann, um wachsenden Anforderungen gerecht zu werden. Kontinuierliche Überwachung, Profiling und Optimierungsbemühungen sind unerlässlich, um die Leistung und Reaktionsfähigkeit des Systems aufrechtzuerhalten, während die Arbeitslast und Komplexität im Laufe der Zeit zunehmen.

Testen und Validierung von Workflows

Testen und Validierung sind kritische Aspekte bei der Entwicklung und Wartung intelligenter Workflow-Orchestrierungssysteme. Angesichts der komplexen Natur KI-gestützter Workflows ist es wichtig sicherzustellen, dass jede Komponente wie erwartet funktioniert, der Gesamtworkflow korrekt arbeitet und die KI-Entscheidungen präzise und zuverlässig sind. In diesem Abschnitt werden wir verschiedene Techniken und Überlegungen für das Testen und Validieren intelligenter Workflows untersuchen.

Unit-Testing von Workflow-Komponenten

Unit-Testing beinhaltet das isolierte Testen einzelner Workflow-Komponenten, um ihre Korrektheit und Robustheit zu überprüfen. Beim Unit-Testing von KI-gestützten Workflow-Komponenten sind folgende Aspekte zu berücksichtigen:

- 1. Eingabevalidierung:** Testen Sie die Fähigkeit der Komponente, verschiedene Arten von Eingaben zu verarbeiten, einschließlich gültiger und ungültiger Daten. Überprüfen Sie, ob die Komponente Grenzfälle angemessen behandelt und entsprechende Fehlermeldungen oder Ausnahmen bereitstellt.
- 2. Ausgabeüberprüfung:** Stellen Sie sicher, dass die Komponente für einen bestimmten Satz von Eingaben die erwarteten Ausgaben erzeugt. Vergleichen Sie die tatsächlichen

Ausgaben mit den erwarteten Ergebnissen, um die Korrektheit zu gewährleisten.

3. Fehlerbehandlung: Testen Sie die Fehlerbehandlungsmechanismen der Komponente, indem Sie verschiedene Fehlerszenarien simulieren, wie ungültige Eingaben, nicht verfügbare Ressourcen oder unerwartete Ausnahmen. Überprüfen Sie, ob die Komponente Fehler angemessen abfängt und behandelt.

4. Randbedingungen: Testen Sie das Verhalten der Komponente unter Randbedingungen, wie leere Eingaben, maximale Eingabegröße oder extreme Werte. Stellen Sie sicher, dass die Komponente diese Bedingungen elegant handhabt, ohne abzustürzen oder falsche Ergebnisse zu produzieren.

Hier ist ein Beispiel eines Unit-Tests für eine Workflow-Komponente in Ruby unter Verwendung des RSpec-Testing-Frameworks:

```
1  RSpec.describe OrderValidator do
2    describe '#validate' do
3      context 'when order is valid' do
4        let(:order) { build(:order) }
5
6        it 'returns true' do
7          expect(subject.validate(order)).to be true
8        end
9      end
10
11     context 'when order is invalid' do
12       let(:order) { build(:order, total_amount: -100) }
13
14       it 'returns false' do
15         expect(subject.validate(order)).to be false
16       end
17     end
18   end
19 end
```

In diesem Beispiel wird die `OrderValidator`-Komponente mit zwei Testfällen getestet: einem für eine gültige Bestellung und einem für eine ungültige Bestellung. Die Testfälle

überprüfen, ob die `validate`-Methode den erwarteten booleschen Wert basierend auf der Gültigkeit der Bestellung zurückgibt.

Integration Testing von Workflow-Interaktionen

Integrationstests konzentrieren sich darauf, die Interaktionen und den Datenfluss zwischen verschiedenen Workflow-Komponenten zu überprüfen. Sie stellen sicher, dass die Komponenten nahtlos zusammenarbeiten und die erwarteten Ergebnisse liefern. Bei der Integration von intelligenten Workflows sollten folgende Aspekte berücksichtigt werden:

- 1. Komponenteninteraktion:** Testen Sie die Kommunikation und den Datenaustausch zwischen Workflow-Komponenten. Überprüfen Sie, ob die Ausgabe einer Komponente korrekt als Eingabe an die nächste Komponente im Workflow weitergegeben wird.
- 2. Datenkonsistenz:** Stellen Sie sicher, dass die Daten während des Durchlaufs durch den Workflow konsistent und genau bleiben. Überprüfen Sie, ob Datentransformationen, Berechnungen und Aggregationen korrekt durchgeführt werden.
- 3. Ausnahmeweiterleitung:** Testen Sie, wie Ausnahmen und Fehler über Workflow-Komponenten weitergeleitet und behandelt werden. Überprüfen Sie, ob Ausnahmen entsprechend erfasst, protokolliert und behandelt werden, um Workflow-Unterbrechungen zu vermeiden.
- 4. Asynchrones Verhalten:** Wenn der Workflow asynchrone Komponenten oder parallele Ausführung beinhaltet, testen Sie die Koordinations- und Synchronisationsmechanismen. Stellen Sie sicher, dass der Workflow sich in gleichzeitigen und asynchronen Szenarien korrekt verhält.

Hier ist ein Beispiel eines Integrationstests für einen Workflow in Ruby unter Verwendung des RSpec-Testframeworks:

```
1 RSpec.describe OrderProcessingWorkflow do
2
3   let(:order) { build(:order) }
4
5   it 'processes the order successfully' do
6     expect(OrderValidator).to receive(:validate).and_return(true)
7     expect(InventoryManager).to receive(:check_availability).and_return(true)
8     expect(PaymentProcessor).to receive(:process_payment).and_return(true)
9     expect(ShippingService).to receive(:schedule_shipping).and_return(true)
10
11     workflow = OrderProcessingWorkflow.new(order)
12     result = workflow.process
13
14     expect(result).to be true
15     expect(order.status).to eq('processed')
16   end
17
18 end
```

In diesem Beispiel wird der `OrderProcessingWorkflow` getestet, indem die Interaktionen zwischen verschiedenen Workflow-Komponenten überprüft werden. Der Testfall legt die Erwartungen für das Verhalten jeder Komponente fest und stellt sicher, dass der Workflow die Bestellung erfolgreich verarbeitet und den Bestellstatus entsprechend aktualisiert.

Testen von KI-Entscheidungspunkten

Das Testen von KI-Entscheidungspunkten ist entscheidend, um die Genauigkeit und Zuverlässigkeit von KI-gesteuerten Workflows sicherzustellen. Beim Testen von KI-Entscheidungspunkten sollten folgende Aspekte berücksichtigt werden:

- 1. Entscheidungsgenauigkeit:** Überprüfen Sie, ob die KI-Komponente auf Grundlage der Eingabedaten und des trainierten Modells akkurate Entscheidungen trifft. Vergleichen Sie die KI-Entscheidungen mit erwarteten Ergebnissen oder Referenzdaten.
- 2. Randfälle:** Testen Sie das Verhalten der KI-Komponente in Randfällen und

ungewöhnlichen Szenarien. Überprüfen Sie, ob die KI-Komponente diese Fälle angemessen behandelt und vernünftige Entscheidungen trifft.

3. Verzerrung und Fairness: Bewerten Sie die KI-Komponente auf mögliche Verzerrungen und stellen Sie sicher, dass sie faire und unvoreingenommene Entscheidungen trifft. Testen Sie die Komponente mit verschiedenartigen Eingabedaten und analysieren Sie die Ergebnisse auf diskriminierende Muster.

4. Erklärbarkeit: Falls die KI-Komponente Erklärungen oder Begründungen für ihre Entscheidungen liefert, überprüfen Sie die Richtigkeit und Klarheit dieser Erklärungen. Stellen Sie sicher, dass die Erklärungen mit dem zugrundeliegenden Entscheidungsprozess übereinstimmen.

Hier ist ein Beispiel für das Testen eines KI-Entscheidungspunkts in Ruby mit dem RSpec Test-Framework:

```
1  RSpec.describe FraudDetector do
2    describe '#detect_fraud' do
3      context 'when transaction is fraudulent' do
4        let(:tx) do
5          build(:transaction, amount: 10_000, location: 'High-Risk Country')
6        end
7
8        it 'returns true' do
9          expect(subject.detect_fraud(tx)).to be true
10        end
11      end
12
13      context 'when transaction is legitimate' do
14        let(:tx) do
15          build(:transaction, amount: 100, location: 'Low-Risk Country')
16        end
17
18        it 'returns false' do
19          expect(subject.detect_fraud(tx)).to be false
20        end
21      end
22    end
23  end
```


In diesem Beispiel wird die KI-Komponente `FraudDetector` mit zwei Testfällen geprüft: einem für eine betrügerische Transaktion und einem für eine legitime Transaktion. Die Testfälle überprüfen, ob die Methode `detect_fraud` basierend auf den Eigenschaften der Transaktion den erwarteten booleschen Wert zurückgibt.

End-to-End-Testing

End-to-End-Testing umfasst das Testen des gesamten Workflows von Anfang bis Ende, wobei reale Szenarien und Benutzerinteraktionen simuliert werden. Es stellt sicher, dass der Workflow sich korrekt verhält und die gewünschten Ergebnisse liefert. Bei der Durchführung von End-to-End-Tests für intelligente Workflows sollten folgende Aspekte berücksichtigt werden:

- 1. Benutzerszenarien:** Identifizieren Sie häufige Benutzerszenarien und testen Sie das Verhalten des Workflows unter diesen Szenarien. Überprüfen Sie, ob der Workflow Benutzereingaben korrekt verarbeitet, angemessene Entscheidungen trifft und die erwarteten Ausgaben erzeugt.
- 2. Datenvalidierung:** Stellen Sie sicher, dass der Workflow Benutzereingaben validiert und bereinigt, um Dateninkonsistenzen oder Sicherheitslücken zu verhindern. Testen Sie den Workflow mit verschiedenen Arten von Eingabedaten, einschließlich gültiger und ungültiger Daten.
- 3. Fehlerbehebung:** Testen Sie die Fähigkeit des Workflows, sich von Fehlern und Ausnahmen zu erholen. Simulieren Sie Fehlerszenarien und überprüfen Sie, ob der Workflow diese angemessen behandelt, die Fehler protokolliert und geeignete Wiederherstellungsmaßnahmen ergreift.
- 4. Leistung und Skalierbarkeit:** Bewerten Sie die Leistung und Skalierbarkeit des Workflows unter verschiedenen Lastbedingungen. Testen Sie den Workflow mit einer großen Anzahl gleichzeitiger Anfragen und messen Sie Antwortzeiten, Ressourcenauslastung und allgemeine Systemstabilität.

Hier ist ein Beispiel eines End-to-End-Tests für einen Workflow in Ruby unter Verwendung des RSpec Test-Frameworks und der Capybara-Bibliothek zur Simulation von Benutzerinteraktionen:

```
1  RSpec.describe 'Order Processing Workflow' do
2    scenario 'User places an order successfully' do
3      visit '/orders/new'
4      fill_in 'Product', with: 'Sample Product'
5      fill_in 'Quantity', with: '2'
6      fill_in 'Shipping Address', with: '123 Main St'
7      click_button 'Place Order'
8
9      expect(page).to have_content('Order Placed Successfully')
10     expect(Order.count).to eq(1)
11     expect(Order.last.status).to eq('processed')
12   end
13 end
```

In diesem Beispiel simuliert der End-to-End-Test einen Benutzer, der eine Bestellung über die Weboberfläche aufgibt. Er füllt die erforderlichen Formularfelder aus, sendet die Bestellung ab und überprüft, ob die Bestellung erfolgreich verarbeitet wird, indem er die entsprechende Bestätigungsnachricht anzeigt und den Bestellstatus in der Datenbank aktualisiert.

Kontinuierliche Integration und Bereitstellung

Um die Zuverlässigkeit und Wartbarkeit intelligenter Workflows sicherzustellen, wird empfohlen, Tests und Validierung in die Pipeline der kontinuierlichen Integration und Bereitstellung (CI/CD) zu integrieren. Dies ermöglicht automatisierte Tests und Validierung von Workflow-Änderungen, bevor diese in Produktion bereitgestellt werden. Berücksichtigen Sie die folgenden Praktiken:

1. **Automatisierte Testausführung:** Konfigurieren Sie die CI/CD-Pipeline so, dass die Testsuite automatisch ausgeführt wird, wenn Änderungen am Workflow-Code

vorgenommen werden. Dies stellt sicher, dass Regressionen oder Fehler früh im Entwicklungsprozess erkannt werden.

2. Überwachung der Testabdeckung: Messen und überwachen Sie die Testabdeckung der Workflow-Komponenten und KI-Entscheidungspunkte. Streben Sie eine hohe Testabdeckung an, um sicherzustellen, dass kritische Pfade und Szenarien gründlich getestet werden.

3. Kontinuierliches Feedback: Integrieren Sie Testergebnisse und Code-Qualitätsmetriken in den Entwicklungsworkflow. Bieten Sie Entwicklern kontinuierliches Feedback über den Status der Tests, die Code-Qualität und eventuelle Probleme, die während des CI/CD-Prozesses erkannt wurden.

4. Staging-Umgebungen: Stellen Sie den Workflow in Staging-Umgebungen bereit, die der Produktionsumgebung sehr ähnlich sind. Führen Sie zusätzliche Tests und Validierungen in der Staging-Umgebung durch, um Probleme im Zusammenhang mit Infrastruktur, Konfiguration oder Datenintegration zu erkennen.

5. Rollback-Mechanismen: Implementieren Sie Rollback-Mechanismen für den Fall von Bereitstellungsfehlern oder kritischen Problemen in der Produktion. Stellen Sie sicher, dass der Workflow schnell auf eine frühere stabile Version zurückgesetzt werden kann, um Ausfallzeiten und Auswirkungen auf die Benutzer zu minimieren.

Durch die Integration von Tests und Validierung während des gesamten Entwicklungslebenszyklus intelligenter Workflows können Organisationen die Zuverlässigkeit, Genauigkeit und Wartbarkeit ihrer KI-gestützten Systeme sicherstellen. Regelmäßige Tests und Validierung helfen dabei, Fehler zu erkennen, Regressionen zu verhindern und das Vertrauen in das Verhalten und die Ergebnisse des Workflows aufzubauen.

Teil 2: Die Muster

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Prompt-Engineering

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Chain of Thought

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Funktionsweise

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Beispiele

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Inhaltserstellung

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Strukturierte Entitätserstellung

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

LLM-Agenten-Anleitung

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Vorteile und Überlegungen

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Moduswechsel

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Funktionsweise

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Wann man es verwendet

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Beispiel

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Rollenzuweisung

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Wie es funktioniert

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Wann man es verwendet

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Beispiele

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Prompt-Objekt

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Funktionsweise

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Prompt-Vorlage

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Funktionsweise

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Vorteile und Überlegungen

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Wann man es verwendet:

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Beispiel

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Structured IO

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Funktionsweise

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Skalierung von Structured IO

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Vorteile und Überlegungen

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Prompt Chaining

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Wie es funktioniert

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Wann es einzusetzen ist

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Beispiel: Olympias Onboarding

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Prompt Rewriter

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Funktionsweise

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Beispiel

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Response Fencing

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Funktionsweise

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Vorteile und Überlegungen

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Fehlerbehandlung

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Query-Analyzer

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Funktionsweise

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Implementierung

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Wortartmarkierung (POS-Tagging) und Eigennamenerkennung (NER)

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Intentionsklassifizierung

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Schlüsselwortextraktion

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Vorteile

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Query Rewriter

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Funktionsweise

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Beispiel

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Vorteile

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Ventriloquist

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Funktionsweise

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Wann es einzusetzen ist

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Beispiel

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Diskrete Komponenten

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Prädikat

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Wie es funktioniert

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Wann man es verwendet

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Beispiel

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

API-Fassade

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Funktionsweise

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Hauptvorteile

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Wann sie einzusetzen ist

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Beispiel

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Authentifizierung und Autorisierung

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Anfragenverarbeitung

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Antwortformatierung

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Fehlerbehandlung und Randfälle

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Skalierbarkeits- und Performanceüberlegungen

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Vergleich mit anderen Entwurfsmustern

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Result Interpreter

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Funktionsweise

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Wann man es einsetzen sollte

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Beispiel

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Virtuelle Maschine

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Wie es funktioniert

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Wann man es verwendet

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Beispiel

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Hinter der Magie

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Spezifikation und Testing

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Verhalten spezifizieren

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Testfälle schreiben

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Beispiel: Testen der Übersetzer-Komponente

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Wiederholung von HTTP-Interaktionen

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Human In The Loop (HITL)

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Übergeordnete Muster

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Hybride Intelligenz

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Adaptive Reaktion

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Mensch-KI-Rollenwechsel

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Eskalation

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Funktionsweise

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Wichtige Vorteile

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Praxisanwendung: Gesundheitswesen

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Feedback-Schleife

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Wie es funktioniert

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Anwendungen und Beispiele

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Fortgeschrittene Techniken der Integration menschlichen Feedbacks

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Passive Informationsabstrahlung

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Funktionsweise

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Kontextuelle Informationsanzeige

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Proaktive Benachrichtigungen

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Erklärende Einblicke

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Interaktive Erkundung

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Wichtige Vorteile

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Anwendungen und Beispiele

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Kollaborative Entscheidungsfindung (CDM)

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Funktionsweise

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Beispiel

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Kontinuierliches Lernen

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Funktionsweise

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Anwendungen und Beispiele

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Beispiel

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Ethische Überlegungen

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Rolle von HITL bei der Minderung von KI-Risiken

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Technologische Fortschritte und Zukunftsausblick

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Herausforderungen und Grenzen von HITL-Systemen

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Intelligente Fehlerbehandlung

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Traditionelle Fehlerbehandlungsansätze

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Kontextuelle Fehlerdiagnose

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Funktionsweise

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Prompt Engineering für kontextuelle Fehlerdiagnose

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Retrieval-Augmented Generation für kontextbezogene Fehlerdiagnose

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Intelligente Fehlerberichterstattung

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Prädiktive Fehlervermeidung

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Funktionsweise

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Intelligente Fehlerbehebung

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Funktionsweise

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Personalisierte Fehlerkommunikation

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Funktionsweise

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Adaptiver Fehlerbehandlungs-Workflow

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Funktionsweise

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Qualitätskontrolle

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Eval

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Problem

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Lösung

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Funktionsweise

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Beispiel

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Überlegungen

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Verständnis von Goldstandards

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Wie referenzfreie Evaluierungen funktionieren

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Schutzmechanismus

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Problem

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Lösung

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Funktionsweise

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Beispiel

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Überlegungen

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Guardrails und Evaluierungen: Zwei Seiten derselben Medaille

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Die Austauschbarkeit von Guardrails und referenzfreien Evaluierungen

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Implementierung von dualen Guardrails und Evaluierungen

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Glossar

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Glossar

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

A

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

B

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

C

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

D

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

E

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

F

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

G

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

H

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

I

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

J

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

K

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

L

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

M

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

N

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

O

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

P

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Q

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

R

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

S

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

T

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

U

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

V

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

W

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Z

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann bei Leanpub unter <http://leanpub.com/patterns-of-application-development-using-ai-de> gekauft werden.

Register

- abrufbasierte Modelle, 7
- ACID-Eigenschaften, 114
- adaptive Benutzeroberfläche, 217
- adaptiver Workflow
 - Adaptive
 - Workflow-Zusammenstellung, 235
- Agentisch, 33
- AI, 77, 102, 140, 149, 211
 - model, 102
- Alpaca, 13
- Altman, Sam, 18
- Amazon Web Services, 262
- Anbieter von
 - Open-Source-Modell-Hosting, 214
- Anpassung, 27
- Anthropic, 23, 40, 76, 134, 142
- anthropomorphismus, 71
- Anwendungsentwicklung, 230
- APIs, 75, 128, 160
- application design and frameworks, 207
- Arrays, 136
- asynchrone Verarbeitung, 259
- Audit-Logging, 110
- Augmented-Reality-Brillen, 228
- Ausnahmebehandlung, 236, 238
- Automatische Fortsetzung, 167
- automatische Skalierung, 262
- autoregressives Modellieren, 44
- Barrierefreiheit, 226, 227
- Basismodelle, 56
- Batch-Verarbeitung, 260
- Befehlszeile
 - Befehlszeilenschnittstelle (CLI), 26
- Benutzbarkeitsprobleme, 225
- Benutzererfahrung, 203
- Benutzeroberfläche (UI)
 - Frameworks, 223
 - Schnittstellen, 223
 - Technologien, 218
- Benutzertests und Feedback, 206
- BERT, 14, 24
- Betrugserkennung
 - System, 100
- Bildungsanwendungen, 33
- boundary conditions, 265
- Brotli, 263
- Byte Pair Encoding (BPE), 13, 15
- C (Programmiersprache), 121
- Caching, 260

- Capybara-Bibliothek, 270
- Chain of Thought (CoT), 46, 144
- Chatbot-Anwendung, 123
- ChatGPT, 31, 55
- Circuit-Breaker-Logik, 169
- classification, 125
- Claude, 8, 45, 80
- Claude 3, 51, 132, 134, 140, 142
- Claude 3 Opus, 78
- Claude v1, 17
- Claude v2, 17
- Cohere (LLM-Anbieter), 23, 25
- computer science, 76
- concurrent workflows, 264
- context
 - Augmentation, 47
 - infinitely long inputs, 16
 - Kontextbasierte Inhaltsgenerierung, 208, 209
 - Kontextbezogene Feldvorschläge, 210
 - window, 16
- Customer Sentiment Analysis, 103
- databases
 - backed object, 109
- Databricks-Mitarbeiter, 54
- Datadog, 258
- Daten
 - Analyse, 35
 - analyse, 153
 - Datenabruf, 113
 - Datenschutz, 27, 225
 - Datensynchronisation, 113
 - Datenvalidierung, 269
 - Fluss, 114
 - Integrität, 250
 - Verarbeitungs-Pipeline, 250
 - Verarbeitungsaufgaben, 131
- Datenbanken, 128
 - Sperrmechanismen, 114
- Datenpersistenz, 113
- Datenstromverarbeitung, 157
- Datenvorbereitung, 113
- Debugging, 235
 - und Testen, 137
- decision
 - making capabilities, 102
- Den Pfad eingrenzen, 39
- den Pfad eingrenzen, 40
- Desktop-Computer, 228
- Destillationsprozess, 79
- deterministisches Verhalten, 60
- Dictionaries, 136
- digitale Landschaft, 202
- document clustering, 125
- Dohan, et al., 45
- Durchsatz, 28
- Dynamische Aufgabenweiterleitung, 233
- dynamische UI-Generierung, 196
- Dynamische Werkzeugauswahl, 136
- E-Commerce, 200, 231
- E-Commerce-Anwendungen, 94
- Effizienz, 232

- Eingabe
 - Prompts, 58
- Eingabeparameter, 133
- ELK-Stack, 115
- emotionaler Ton, 151
- End-to-End-Testing, 269
- End-to-End-Tests, 270
- Engpässe, 235
- Ensembles, 122, 123
 - Ensemble von Workern, 123
- Enterprise Integration Patterns, 108
- Entscheidung
 - bäume, 231
 - punkte, 255
- Entscheidungsfindung
 - sfälle, 138
- Entwicklungsframeworks, 155
- ereignisgesteuerte Architektur, 112
- Erfassung der Krankengeschichte, 104
- Ergebnisinterpret, 148
- Erklärbarkeit, 268
- errors
 - handling, 265
 - Intelligent Error Handling, 149
- Erzwungene Werkzeugauswahl, 137
- Ethik
 - Implikationen, 208
- Experimentieren
 - Framework, 203
- externe Dienste oder APIs, 131
- F#, 96
- Facebook, 25
- Fallback-Strategien, 114
- Feedback
 - Feedback-Schleife, 61
- Fehler
 - Behandlung, 114, 148
 - Fehlerbehebung, 269
 - Raten, 114
- Fehlerbehandlung, 111
- Fehlersuche
 - und Fehlerbehebung, 257
- few-shot
 - learning, 64
 - prompting, 66
- Finalisierungsmethode, 163
- finalize-Methode, 165, 166
- fine-tuning, 83
- FitAI, 220
- Flexibilität und Kreativität, 205
- Frage-Antwort-Systeme, 7
- Funktion
 - Aufruf, 128, 129, 164
 - Aufrufsverlauf, 163
- funktionale Programmierung, 95
- Funktionen
 - Namen, 161
- Funktionsaufruf
 - Fehlschlag, 139
- Gemma 7B, 11
- Generative Pre-trained Transformer (GPT),
 - 8, 70

- Generative UI (GenUI), 207, 214, 215, 219, 223, 227
- geschlossene und offene
 - Fragebeantwortung, 55
- Geschäftsregeln, 231
- Gespräch
 - Protokoll, 166
 - Schleife, 167
- Gesprächsprotokoll, 164
- GitLab, 96
- Global Interpreter Lock (GIL), 120
- Google, 23
 - API, 65, 68
 - Cloud AI Platform, 25
 - Cloud Platform, 262
 - Gemini, 22
 - Gemini 1.5 Pro, 14, 17, 19
 - PaLM (Pathways Language Model), 18, 24
 - T5, 14
- GPT-3, 13, 17
- GPT-4, 6, 13, 17, 18, 22, 32, 45, 51, 65, 109, 122, 125, 132, 139, 213, 260
- grafische Modelle, 45
- Graham, Paul, 19
- Grammatikregeln, 4
- granulare Protokollierung, 258
- GraphQL, 112
- Groq, 27, 125
- Großes Sprachmodell (GSM), 242
- Großes Sprachmodell (LLM), 18, 30, 79, 146, 213, 218
- Landschaft, 28
- gzip, 263
- Hardware, 29
- Hash, 159
- historische Muster, 235
- Hochleistungs-Textgenerierung, 27
- Hohpe, Gregor, 108
- Honeybadger, 97
- HTTP, 157
- Human-In-The-Loop (HITL), 187
- hyperparameter, 48
- Inferenz, 5
- Informatik, 73
- Information
 - Abruf, 7, 131
 - Extraktion, 55
- Inhalt
 - Filterung, 27
 - Inhaltskategorisierung, 116
- inhaltsbasiertes Filtern, 95
- inklusive Schnittstellen, 208
- input
 - validation, 264
- instruction tuning
 - instruktionsoptimierte Modelle, 54
- Instruktions-Feinabstimmung, 10
- Instruktionsoptimierung
 - instruktionsoptimierte Modelle, 51
- Integration von LLMs, 196
- Integrationstests, 266
- intelligent workflow orchestration, 264

- intelligente Workflow-Orchestrierung, 230, 238, 261
- Intelligenter Inhaltsmoderator, 243
- Internationalisierung, 204
- iterative refinement, 150
- iterative Verfeinerung, 79
- JSON (JavaScript Object Notation), 132, 136, 140, 154, 174
- K-means, 127
- KI, 67, 133, 157, 219
 - Anwendungen, 130, 144, 156, 170
 - Entscheidungspunkte, 267
 - konversationelle, 7
 - Konversations, 32
 - konversationsbasiert, 221
 - Modell, 92, 162, 163, 166, 220
 - zusammengesetzte Systeme, 31, 32, 35
- Klassifizierung, 55
- Klinische Entscheidungsunterstützung, 107
- kollaboratives Filtern, 95
- komplexe Aufgaben, 152
- Konsistenz
 - und Reproduzierbarkeit, 138
- Kontext
 - Fenster, 235
 - kontextbezogene
 - Entscheidungsfindung, 235
 - Kontextbezogene Inhaltsgenerierung, 195
 - Kontextuelle Inhaltsgenerierung, 200–202
- Kontinuierliche Integration und Bereitstellung (CI/CD), 270
 - Pipeline, 270
- Kontinuierliche Risikoüberwachung, 106
- Konto, 94
- Konversation
 - Schleife, 164
- konzeptionelle und praktische Herausforderungen, 208
- kreatives Schreiben, 35, 55
- Kundenservice, 33
- Kundenservice-Chatbots, 34
- language
 - models, 68, 76
- Large Language Model (LLM), 1, 3, 16, 70, 71, 74, 75, 81, 90, 115, 125, 128, 130, 140, 150, 153, 171, 174, 195, 207
- Latente Dirichlet Allocation, 127
- latenter Raum, 41, 43
- Latenz, 28
- Leistung
 - Kompromisse, 5
 - Optimierung, 138, 205, 257
 - Probleme, 261
- Lieferkette
 - Optimierung, 34
- lineare Algebra, 44
- lineare Regression, 45
- Llama, 13
- Llama 2-70B, 52

- Llama 3 70B, 11
- Llama 3 8B, 11
- lokale Entwicklungsumgebungen, 162
- Louvre, 44
- Managed Streaming for Apache Kafka, 42
- manuelle Intervention, 238
- Markdown, 153
- Markup-Style-Tagging, 74
- medizinische Entdeckungen, 104
- Mehrheitsentscheidung, 122
- mehrstufiger Workflow, 115
- Memorial Sloan Kettering Cancer Center, 42
- Merkur (Planet), 46
- Merkur (römischer Gott), 46
- MessagePack, 263
- Meta, 25
- Metropolitan Museum of Art, 44
- Microservices-Architektur, 92
- Mistral, 26
 - 7B, 11
 - 7B Instruct, 17, 213
- Mixtral
 - 8x22B, 11
 - 8x7B, 58
- modalitätsübergreifende Generierung, 22
- moderne Anwendungen, 233
- Modularität, 91
- Motivationsstrategien, 223
- Multi-Agent
 - Problemlöser, 32
- Multimodal
 - language models, 21
 - models, 20
- Multitude of Workers, 173
- Mustererkennung, 159
- Naive Bayes, 126
- narrative building, 20
- natural language
 - Verarbeitung natürlicher Sprache (NLP), 125
- natürliche Sprache
 - Verarbeitung natürlicher Sprache (NLP), 104
- Netzwerkonnektivität, 236
- neuronale Netze, 4
- neuronale Netzwerke, 6
- New Relic, 261
- Notfalleinsatzplanung, 34
- nutzergenerierte Inhalte, 116
- Nutzerpsychologie, 224
- Nutzervertrauen, 226
- Ollama, 26
- Olympia, 34, 65, 133, 149, 158, 174
- Olympias Wissensdatenbank, 94
- One-Shot-Learning, 63
- Online-Händler, 214
- OpenAI, 3, 23, 40, 76
- OpenRouter, 28, 29, 158, 261
- OPT model, 25
- optimistisches Sperren, 114
- output verification, 265

- parallele Ausführung, 260
- Parameter
 - Auswirkungen, 134
 - Bereich, 11
 - Parameteranzahl, 29
- Paraphrasierung, 55
- Perplexity (Anbieter), 12
- personalisierte Produktempfehlungen, 95
- Personalisierung, 197, 227, 232
 - Personalisierte Formulare, 209
 - Personalisierte Microcopy, 215
- pessimistisches Sperren, 114
- principle of least privilege, 75
- probabilistische Modelle, 44
- Process Manager, 108, 111
 - Enterprise Integration, 239
- Produktempfehlungen, 95
- Produktivität, 198
- Progressive Disclosure, 216
- Prompts
 - Design, 60
 - Engineering, 42, 58, 62, 224
 - Prompt Template, 214
 - Prompt-Destillation, 260
 - Prompt-Vorlage, 62
 - Verkettung, 61
- prompts
 - chaining, 74
 - design, 71
 - Engineering, 68
 - engineering, 46, 47, 70
 - Prompt Distillation, 47, 76
 - Prompt Object, 77
 - Prompt-Destillation, 81
 - refinement, 71
- Protocol Buffers, 263
- Protokollaufbewahrung und -rotation, 258
- Präsenzstrafe, 50
- Prüfung und Compliance, 257
- Publish-Subscribe-Systeme, 112
- PyTorch, 25
- Quantisierung, 29
- Quecksilber (Element), 46
- Qwen2 70B, 11
- Rails, 204
- Railway Oriented Programming (ROP), 98
- Raix, 239
 - Bibliothek, 100
- Randfälle, 60
- Rangiersysteme, 37
- Response Fencing, 184, 214
- Retrieval Augmented Generation (RAG),
 - 32, 39, 47, 83, 130
- Risikofaktoren, 99, 100
- Risikostratifizierung, 106
- Rollback-Mechanismen, 271
- Rollenspiel-artige Interaktionen, 7
- RSpec, 265, 266, 270
- Ruby, 96, 97, 117, 170, 270
- Ruby on Rails, 1, 116, 239, 247
- Rudall, Alex, 24
- Rust (Programmiersprache), 121
- Rust (Programming Language), 96

- Scout, 261
- Segmentierungs- und Targeting-Strategien, 203
- Selbstheilende Daten, 171, 253
- Sentimentanalyse, 118
- serverseitige Ereignisse (SSE), 157
- Skalierbarkeit, 232, 259
- Smartphones, 228
- Softwarearchitektur, 2
- Sprache
 - bezogene Aufgaben, 5
 - modelle, 44
 - Spracherkennung, 116
- sprachgesteuerte Schnittstellen, 34
- SQL-Injektionen, 73
- Staging-Umgebungen, 271
- Stimmungsanalyse, 17, 103, 116, 117, 122, 123, 140, 151
- Stream-Handler, 158
- Stream-Verarbeitung, 170
- Streaming-Daten, 159
- Streamverarbeitung, 163
 - Logik, 165
- Stripe, 134
- Structured IO, 214
- strukturierte Daten, 139
- strukturierte Protokollierung, 258
- Support Vector Machines (SVM), 126
- Symptombeurteilung und Stratifizierung, 104
- Syntaxfehler, 136
- synthetische Datengenerierung, 55
- system directive, 102
- Systemdirektive, 133
- T5, 24
- Tablets, 228
- Temperatur, 56
- Textbereinigung, 116
- Theory of Mind, 41
- Ticket-Zuweisung, 249
- Together.ai, 27
- Token, 6
- Tokenisierung, 13
- Tokens, 13
- Top-k Sampling, 50
- Top-p (Nucleus) Sampling, 49
- topic identification, 125
- Tragik der Allmende, 200
- Trainingsdaten, 43
- Transformer-Architektur, 6
- trigger message, 108
- Unicode-kodierbare Sprache, 15
- Universal ID, 263
- Unternehmensanwendungsarchitektur, 39
- unüberwachtes Lernen, 4
- User Interface (UI)
 - Benutzeroberflächen, 207
 - Design, 228
- Ventriloquist, 184
- Verarbeitungszeit, 114
- Verfolgung wichtiger Metriken, 254
- Verkehrsmanagement, 34

- Verkettung von KI-Workern, 115
- Versicherungsüberprüfung, 105
- verteilte Architektur, 259
- Verzerrung
 - und Fairness in KI, 268
- Vielzahl von Workern, 123
- virtuelle Assistenten, 34
- visuelle Schnittstelle, 218
- Vorhersagen, 6
- Wall, Larry, 3
- Werkzeugaufruf, 160
- Werkzeugeinsatz, 155
- Werkzeugnutzung, 128, 129
- wichtige Muster, 233
- Wiederholungsmechanismen, 114
- Wiederholungsstrafen, 53
- Wisper, 97, 110, 158, 165
- Wissensdatenbanken, 7
- Wissensmanagement, 33
- Wooley, Chad, 96
- XML, 140
- Yi-34B, 52
- Zeit bis zum ersten Token (TTFT), 28
- Zero-Shot-Learning, 62
- Zero-Shot-Lernen, 61
- Zusammenfassung, 55
- zustandslos, 164
- Ökosystem, 154
- Übersetzung, 17, 205
- Überwachung
 - Metriken, 257
 - und Alarmierung, 237
 - und Protokollierung, 114, 256