

[Produkt](#)

# Aufbau wirksamer Mittel

19. Dez. 2024

Im vergangenen Jahr haben wir mit Dutzenden von Teams zusammengearbeitet, um Large Language Model (LLM)-Agenten in verschiedenen Branchen aufzubauen. Die erfolgreichsten Implementierungen wurden durchweg nicht mit komplexen Frameworks oder spezialisierten Bibliotheken durchgeführt. Stattdessen bauten sie mit einfachen, zusammensetzbaren Mustern.

In diesem Beitrag teilen wir mit, was wir aus der Zusammenarbeit mit unseren Kunden und Baumaklern gelernt haben, und geben Entwicklern praktische Ratschläge zum Aufbau effektiver Agenten.

## Was sind Agenten?

"Agent" kann auf verschiedene Arten definiert werden. Einige Kunden definieren Agenten als vollständig autonome Systeme, die über längere Zeiträume unabhängig voneinander arbeiten und verschiedene Tools verwenden, um komplexe Aufgaben zu erledigen. Andere verwenden den Begriff, um präskriptive Implementierungen zu beschreiben, die vordefinierten Workflows folgen. Bei Anthropic kategorisieren wir all diese Variationen als **agentische Systeme**, treffen aber eine wichtige architektonische Unterscheidung zwischen **Workflows** und **Agenten**:

- **Workflows** sind Systeme, in denen LLMs und Tools über vordefinierte Codepfade orchestriert werden.
- **Agenten** hingegen sind Systeme, in denen LLMs ihre eigenen Prozesse und die Nutzung von Tools dynamisch steuern und so die Kontrolle darüber behalten, wie sie Aufgaben erledigen.

Im Folgenden werden wir beide Arten von agentischen Systemen im Detail untersuchen. In Anhang 1 ("Agenten in der Praxis") beschreiben wir zwei Bereiche, in denen Kunden besonderen Wert auf die Nutzung solcher Systeme gelegt haben.

## **Wann (und wann nicht) Agenten verwendet werden sollten**

Bei der Erstellung von Anwendungen mit LLMs empfehlen wir, die einfachste Lösung zu finden und die Komplexität nur bei Bedarf zu erhöhen. Das könnte bedeuten, dass überhaupt keine agentischen Systeme gebaut werden. Agentische Systeme tauschen oft Latenz und Kosten gegen eine bessere Aufgabenleistung ein, und Sie sollten überlegen, wann dieser Kompromiss sinnvoll ist.

Wenn mehr Komplexität erforderlich ist, bieten Workflows Vorhersagbarkeit und Konsistenz für klar definierte Aufgaben, während Agenten die bessere Option sind, wenn Flexibilität und modellgesteuerte Entscheidungsfindung in großem Maßstab erforderlich sind. Für viele Anwendungen reicht es jedoch in der Regel aus, einzelne LLM-Aufrufe mit Retrieval und In-Context-Beispielen zu optimieren.

## **Wann und wie man Frameworks verwendet**

Es gibt viele Frameworks, die die Implementierung von agentischen Systemen erleichtern, darunter:

- LangGraph von LangChain;
- Das KI-Agenten-Framework von Amazon Bedrock;
- Rivet, ein Drag-and-Drop-GUI-LLM-Workflow-Builder; und
- Vellum, ein weiteres GUI-Tool zum Erstellen und Testen komplexer Workflows.

Diese Frameworks erleichtern den Einstieg, indem sie standardmäßige Low-Level-Aufgaben wie das Aufrufen von LLMs, das Definieren und Parsen von Tools und das Verketten von Aufrufen vereinfachen. Sie erstellen jedoch häufig zusätzliche Abstraktionsebenen, die die zugrunde liegenden

Eingabeaufforderungen und Antworten verschleiern können, wodurch sie schwieriger zu debuggen sind. Sie können es auch verlockend machen, die Komplexität zu erhöhen, wenn eine einfachere Einrichtung ausreichen würde.

Wir empfehlen Entwicklern, mit der direkten Verwendung von LLM-APIs zu beginnen: Viele Muster können in wenigen Codezeilen implementiert werden. Wenn Sie ein Framework verwenden, stellen Sie sicher, dass Sie den zugrunde liegenden Code verstehen. Falsche Annahmen darüber, was unter der Haube steckt, sind eine häufige Quelle für Kundenfehler.

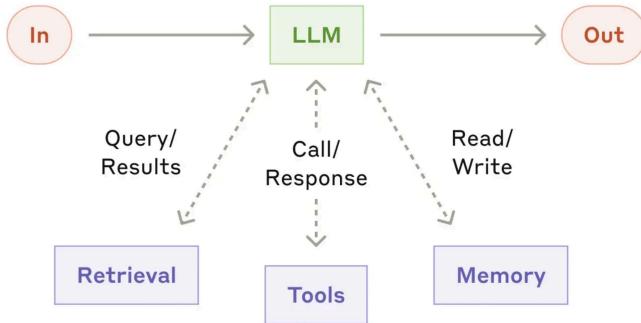
In unserem [Rezeptbuch](#) finden Sie einige Beispielimplementierungen.

## Bausteine, Workflows und Agenten

In diesem Abschnitt untersuchen wir die gängigen Muster für agentische Systeme, die wir in der Produktion gesehen haben. Wir beginnen mit unserem grundlegenden Baustein – dem erweiterten LLM – und erhöhen schrittweise die Komplexität, von einfachen kompositorischen Workflows bis hin zu autonomen Agenten.

### Baustein: Das erweiterte LLM

Der Grundbaustein agentischer Systeme ist ein LLM, das um Erweiterungen wie Abruf, Werkzeuge und Speicher erweitert wurde. Unsere aktuellen Modelle können diese Funktionen aktiv nutzen, indem sie ihre eigenen Suchanfragen generieren, geeignete Tools auswählen und bestimmen, welche Informationen aufbewahrt werden sollen.



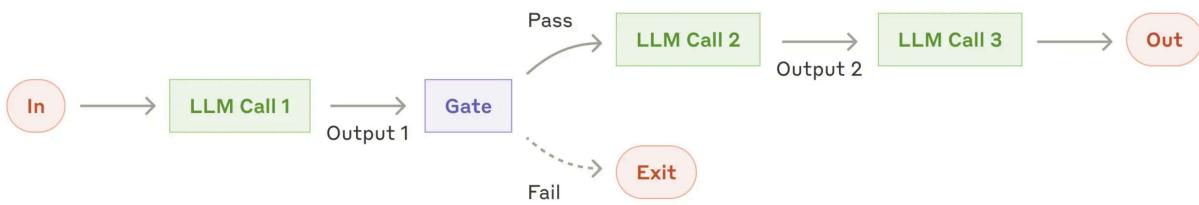
Das erweiterte LLM

Wir empfehlen, sich auf zwei Schlüsselaspekte der Implementierung zu konzentrieren: diese Funktionen auf Ihren spezifischen Anwendungsfall zuzuschneiden und sicherzustellen, dass sie eine einfache, gut dokumentierte Schnittstelle für Ihr LLM bieten. Es gibt zwar viele Möglichkeiten, diese Erweiterungen zu implementieren, aber ein Ansatz ist unser kürzlich veröffentlichtes [Model Context Protocol](#), das es Entwicklern ermöglicht, sich mit einer einfachen [Client-Implementierung](#) in ein wachsendes Ökosystem von Tools von Drittanbietern zu integrieren.

Für den Rest dieses Beitrags gehen wir davon aus, dass jeder LLM-Aufruf Zugriff auf diese erweiterten Funktionen hat.

## Workflow: Verkettung von Eingabeaufforderungen

Bei der Eingabeaufforderungsverkettung wird eine Aufgabe in eine Abfolge von Schritten zerlegt, wobei jeder LLM-Aufruf die Ausgabe des vorherigen Aufrufs verarbeitet. Sie können programmatische Prüfungen (siehe "Gate" im Diagramm unten) für alle Zwischenschritte hinzufügen, um sicherzustellen, dass der Prozess weiterhin auf Kurs ist.



Der Workflow für die Verkettung von Eingabeaufforderungen

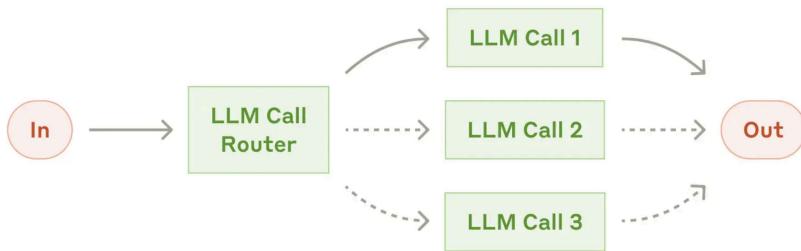
**Wann sollte dieser Workflow verwendet werden:** Dieser Workflow ist ideal für Situationen, in denen die Aufgabe einfach und sauber in feste Unteraufgaben zerlegt werden kann. Das Hauptziel besteht darin, die Latenz gegen eine höhere Genauigkeit einzutauschen, indem jeder LLM-Aufruf zu einer einfacheren Aufgabe wird.

**Beispiele, bei denen die Verkettung von Eingabeaufforderungen nützlich ist:**

- Generierung von Marketingtexten und anschließende Übersetzung in eine andere Sprache.
- Schreiben einer Gliederung eines Dokuments, Überprüfen, ob die Gliederung bestimmte Kriterien erfüllt, und dann das Schreiben des Dokuments auf der Grundlage der Gliederung.

## Arbeitsablauf: Arbeitsplan

Beim Routing wird eine Eingabe klassifiziert und an eine spezialisierte Folgeaufgabe weitergeleitet. Dieser Workflow ermöglicht das Trennen von Bedenken und das Erstellen speziellerer Eingabeaufforderungen. Ohne diesen Workflow kann die Optimierung für eine Art von Eingabe die Leistung anderer Eingaben beeinträchtigen.



Der Routing-Workflow

**Wann sollte dieser Workflow verwendet werden:** Das Routing eignet sich gut für komplexe Aufgaben, bei denen es unterschiedliche Kategorien gibt, die besser getrennt behandelt werden sollten, und bei denen die Klassifizierung genau gehandhabt werden kann, entweder durch ein LLM oder ein traditionelleres Klassifizierungsmodell/einen traditionelleren Klassifizierungsalgorithmus.

**Beispiele, bei denen Routing nützlich ist:**

- Leiten Sie verschiedene Arten von Kundendienstanfragen (allgemeine Fragen, Rückerstattungsanträge, technischer Support) an verschiedene nachgelagerte Prozesse, Eingabeaufforderungen und Tools weiter.
- Weiterleiten von einfachen/häufigen Fragen an kleinere Modelle wie Claude 3.5 Haiku und schwierige/ungewöhnliche Fragen an leistungsfähigere Modelle wie Claude 3.5 Sonnet, um Kosten und Geschwindigkeit zu optimieren.

## Arbeitsablauf: Parallelisierung

LLMs können manchmal gleichzeitig an einer Aufgabe arbeiten und ihre Ausgaben programmgesteuert aggregiert werden. Dieser Workflow, die Parallelisierung, manifestiert sich in zwei wichtigen Varianten:

- **Abschnittierung:** Unterteilen einer Aufgabe in unabhängige Unteraufgaben, die parallel ausgeführt werden.
- **Stimmabgabe:** Führen Sie dieselbe Aufgabe mehrmals aus, um unterschiedliche Ausgaben zu erhalten.



Der Parallelisierungs-Workflow

**Wann sollte dieser Workflow verwendet werden:** Die Parallelisierung ist effektiv, wenn die geteilten Teilaufgaben aus Gründen der Geschwindigkeit parallelisiert werden können oder wenn mehrere Perspektiven oder Versuche erforderlich sind, um Ergebnisse mit höherer Zuverlässigkeit zu erzielen. Bei komplexen Aufgaben mit mehreren Überlegungen schneiden LLMs in der Regel besser ab, wenn jede Überlegung von einem separaten LLM-Aufruf behandelt wird, sodass die Aufmerksamkeit auf jeden spezifischen Aspekt gerichtet werden kann.

**Beispiele, bei denen Parallelisierung nützlich ist:**

- **Einteilung:**
  - Implementieren von Leitplanken, bei denen eine Modellinstanz Benutzerabfragen verarbeitet, während eine andere sie auf unangemessene Inhalte oder Anforderungen überprüft. Dies ist tendenziell leistungsfähiger, als wenn derselbe LLM-Aufruf sowohl die Leitplanken als auch die Kernantwort behandelt.
  - Automatisieren von Auswertungen zur Bewertung der LLM-Leistung, wobei jeder LLM-Aufruf einen anderen Aspekt der Leistung des Modells an einer bestimmten Eingabeaufforderung auswertet.
  
- **Abstimmung:**
  - Überprüfung eines Codeabschnitts auf Schwachstellen, bei der der Code von mehreren verschiedenen Eingabeaufforderungen überprüft und gekennzeichnet wird, wenn ein Problem gefunden wird.
  - Bewertung, ob ein bestimmter Inhalt unangemessen ist, wobei mehrere Eingabeaufforderungen verschiedene Aspekte bewerten oder unterschiedliche Abstimmungsschwellenwerte erfordern, um falsch positive und negative Ergebnisse auszugleichen.

## Workflow: Orchestrator-Worker

Im Orchestrator-Worker-Workflow unterteilt ein zentrales LLM Aufgaben dynamisch, delegiert sie an Worker-LLMs und synthetisiert deren Ergebnisse.



Der Workflow zwischen Orchestrator und Mitarbeitern

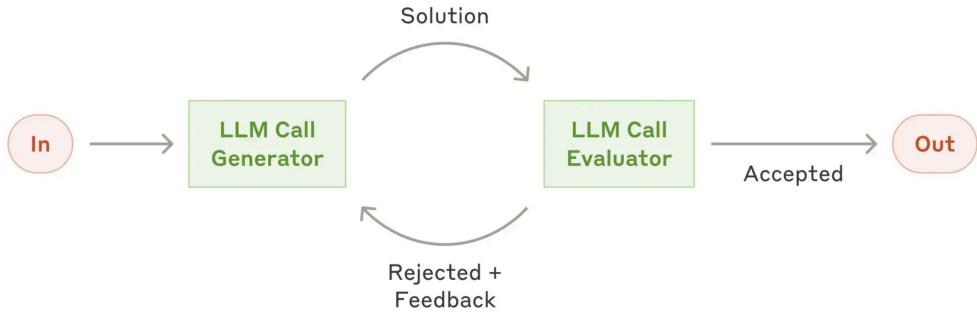
**Wann sollte dieser Workflow verwendet werden:** Dieser Workflow eignet sich gut für komplexe Aufgaben, bei denen Sie die erforderlichen Unteraufgaben nicht vorhersagen können (beim Codieren hängen z. B. die Anzahl der Dateien, die geändert werden müssen, und die Art der Änderung in den einzelnen Dateien wahrscheinlich von der Aufgabe ab). Obwohl es topografisch ähnlich ist, ist der Hauptunterschied zur Parallelisierung die Flexibilität – Unteraufgaben sind nicht vordefiniert, sondern werden vom Orchestrator auf der Grundlage der spezifischen Eingabe bestimmt.

**Beispiel, wo orchestrator-worker nützlich ist:**

- Codierung von Produkten, die jedes Mal komplexe Änderungen an mehreren Dateien vornehmen.
- Suchaufgaben, bei denen Informationen aus mehreren Quellen gesammelt und analysiert werden, um mögliche relevante Informationen zu erhalten.

## Arbeitsablauf: Evaluator-Optimierer

Im Evaluator-Optimizer-Workflow generiert ein LLM-Aufruf eine Antwort, während ein anderer Evaluierung und Feedback in einer Schleife bereitstellt.



Der Arbeitsablauf für Auswerter und Optimierer

**Wann sollte dieser Workflow verwendet werden:** Dieser Workflow ist besonders effektiv, wenn wir klare Bewertungskriterien haben und wenn die iterative Verfeinerung einen messbaren Mehrwert bietet. Die beiden Anzeichen für eine gute Anpassung sind: Erstens, dass LLM-Antworten nachweislich verbessert werden können, wenn ein Mensch sein Feedback artikuliert; und zweitens, dass das LLM ein solches Feedback geben kann. Dies ist analog zum iterativen Schreibprozess, den ein menschlicher Autor durchlaufen kann, wenn er ein ausgereiftes Dokument erstellt.

**Beispiele, bei denen der Evaluator-Optimierer nützlich ist:**

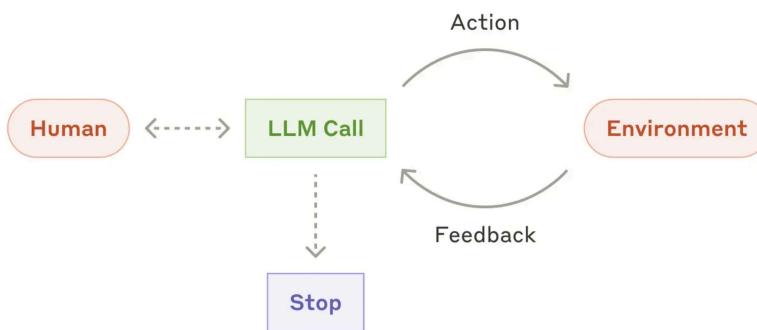
- Literarische Übersetzung, bei der es Nuancen gibt, die der LLM-Übersetzer zunächst nicht erfasst, bei denen aber ein LLM-Bewerter nützliche Kritik üben kann.
- Komplexe Suchaufgaben, die mehrere Such- und Analyserunden erfordern, um umfassende Informationen zu sammeln, bei denen der Bewerter entscheidet, ob weitere Suchen gerechtfertigt sind.

## Agenten

Agenten tauchen in der Produktion auf, da LLMs in Schlüsselfunktionen reifen – komplexe Eingaben verstehen, argumentieren und planen, Tools zuverlässig nutzen und sich von Fehlern erholen. Agenten beginnen ihre Arbeit entweder mit einem Befehl des menschlichen Benutzers oder einer interaktiven Diskussion mit ihm. Sobald die Aufgabe klar ist, planen und arbeiten die Agenten unabhängig voneinander und kehren möglicherweise zum Menschen zurück, um weitere Informationen oder Beurteilungen zu erhalten. Während der

Ausführung ist es für die Agenten von entscheidender Bedeutung, bei jedem Schritt (z. B. bei den Ergebnissen von Tool-Aufrufen oder der Codeausführung) die "Grundwahrheit" aus der Umgebung zu erhalten, um den Fortschritt zu bewerten. Agenten können dann an Kontrollpunkten oder bei der Begegnung mit Blockern eine Pause einlegen, um menschliches Feedback zu erhalten. Die Aufgabe wird häufig nach Abschluss beendet, aber es ist auch üblich, Stoppbedingungen (z. B. eine maximale Anzahl von Iterationen) einzuschließen, um die Kontrolle zu behalten.

Agenten können komplexe Aufgaben bewältigen, aber ihre Implementierung ist oft unkompliziert. Dabei handelt es sich in der Regel nur um LLMs, die Tools verwenden, die auf Umgebungsfeedback in einer Schleife basieren. Daher ist es wichtig, Toolsets und deren Dokumentation klar und durchdacht zu gestalten. Wir erläutern die Best Practices für die Tool-Entwicklung in Anhang 2 ("Prompt Engineering your Tools").



Autonomer Agent

**Wann sollten Agenten verwendet werden:** Agents können für offene Probleme verwendet werden, bei denen es schwierig oder unmöglich ist, die erforderliche Anzahl von Schritten vorherzusagen, und bei denen Sie keinen festen Pfad fest codieren können. Das LLM wird möglicherweise viele Runden lang funktionieren, und Sie müssen ein gewisses Maß an Vertrauen in seine Entscheidungsfindung haben. Die Autonomie der Agenten macht sie ideal für die Skalierung von Aufgaben in vertrauenswürdigen Umgebungen.

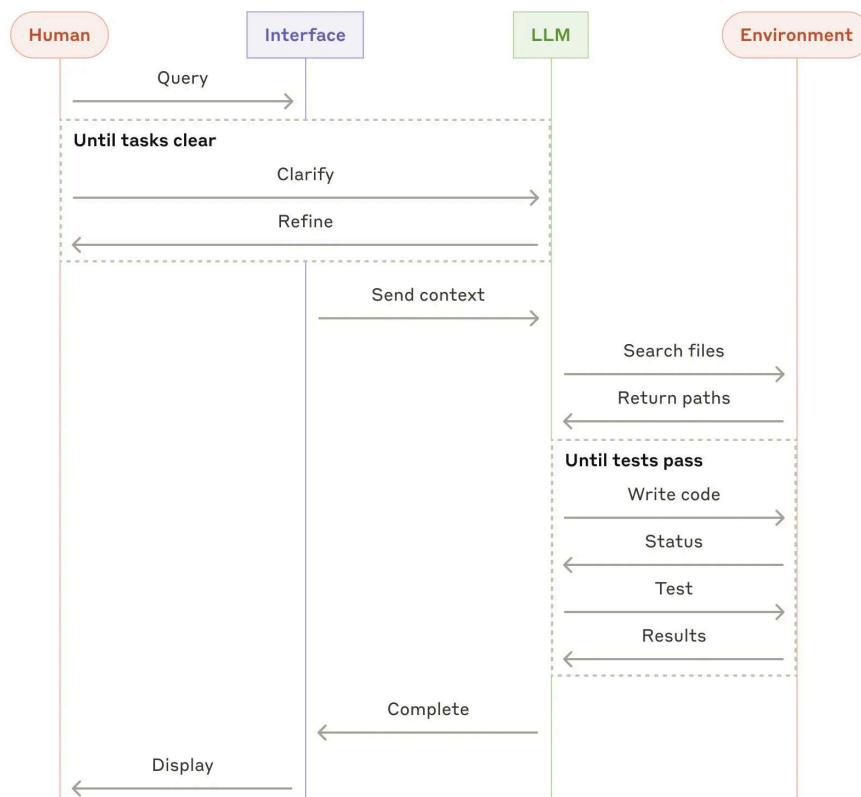
Die Autonomie der Agenten bedeutet höhere Kosten und das Potenzial, Fehler zu verschlimmern. Wir empfehlen umfangreiche Tests in Sandbox-Umgebungen

zusammen mit den entsprechenden Leitplanken.

Beispiele, bei denen Agenten nützlich sind:

Die folgenden Beispiele stammen aus unseren eigenen Implementierungen:

- Ein Codierungsagent zum Lösen von SWE-Bench-Aufgaben, die Änderungen an vielen Dateien auf der Grundlage einer Aufgabenbeschreibung beinhalten;
- Unsere Referenzimplementierung "Computernutzung", bei der Claude einen Computer verwendet, um Aufgaben zu erledigen.



High-Level-Fluss eines Codieragenten

## Kombinieren und Anpassen dieser Muster

Diese Bausteine sind nicht vorgeschrieben. Dabei handelt es sich um gängige Muster, die Entwickler für verschiedene Anwendungsfälle gestalten und kombinieren können. Der Schlüssel zum Erfolg liegt, wie bei allen LLM-Funktionen, in der Messung der Leistung und der Iteration der Implementierungen. Um es noch einmal zu wiederholen: Sie sollten nur dann in

Betracht *ziehen*, die Komplexität zu erhöhen, wenn dies nachweislich die Ergebnisse verbessert.

## Zusammenfassung

Erfolg im LLM-Bereich hängt nicht davon ab, das anspruchsvollste System zu entwickeln. Es geht darum, das *richtige* System für Ihre Bedürfnisse zu entwickeln. Beginnen Sie mit einfachen Eingabeaufforderungen, optimieren Sie diese mit einer umfassenden Auswertung und fügen Sie mehrstufige agentische Systeme nur dann hinzu, wenn einfachere Lösungen nicht ausreichen.

Bei der Implementierung von Agenten versuchen wir, drei Kernprinzipien zu befolgen:

1. Bewahren Sie die **Einfachheit** des Designs Ihres Agenten.
2. Priorisieren Sie **Transparenz**, indem Sie die Planungsschritte des Agenten explizit anzeigen.
3. Erstellen Sie Ihre Agent-Computer-Schnittstelle (ACI) sorgfältig durch **gründliche Tool-Dokumentation und -Tests**.

Frameworks können Ihnen helfen, schnell loszulegen, aber zögern Sie nicht, Abstraktionsschichten zu reduzieren und mit grundlegenden Komponenten zu erstellen, wenn Sie in die Produktion übergehen. Wenn Sie diese Prinzipien befolgen, können Sie Agenten erstellen, die nicht nur leistungsstark, sondern auch zuverlässig, wartbar und von ihren Benutzern vertrauenswürdig sind.

## Bestätigungen

Geschrieben von Erik Schluntz und Barry Zhang. Diese Arbeit stützt sich auf unsere Erfahrungen bei Anthropic und die wertvollen Erkenntnisse unserer Kunden, für die wir zutiefst dankbar sind.

## Anlage 1: Arbeitsstoffe in der Praxis

Unsere Arbeit mit Kunden hat zwei besonders vielversprechende Anwendungen für KI-Agenten aufgedeckt, die den praktischen Wert der oben diskutierten Muster demonstrieren. Beide Anwendungen veranschaulichen, wie Agenten den

größten Mehrwert für Aufgaben schaffen, die sowohl Gespräche als auch Maßnahmen erfordern, klare Erfolgskriterien haben, Feedbackschleifen ermöglichen und eine sinnvolle menschliche Aufsicht integrieren.

## A. Kundensupport

Der Kundensupport kombiniert vertraute Chatbot-Schnittstellen mit erweiterten Funktionen durch Tool-Integration. Dies ist eine natürliche Ergänzung für offenere Agenten, weil:

- Support-Interaktionen folgen auf natürliche Weise einem Gesprächsfluss, erfordern jedoch Zugriff auf externe Informationen und Aktionen.
- Tools können integriert werden, um Kundendaten, Bestellhistorie und Wissensdatenbankartikel abzurufen.
- Aktionen wie das Ausstellen von Rückerstattungen oder das Aktualisieren von Tickets können programmgesteuert gehandhabt werden. und
- Der Erfolg lässt sich durch benutzerdefinierte Auflösungen eindeutig messen.

Mehrere Unternehmen haben die Praktikabilität dieses Ansatzes durch nutzungsbasierte Preismodelle demonstriert, die nur für erfolgreiche Lösungen Gebühren erheben, was das Vertrauen in die Effektivität ihrer Agenten zeigt.

## B. Codieragenten

Der Bereich der Softwareentwicklung hat ein bemerkenswertes Potenzial für LLM-Funktionen gezeigt, wobei sich die Fähigkeiten von der Code-Vervollständigung bis zur autonomen Problemlösung entwickelt haben. Wirkstoffe sind besonders wirksam, weil:

- Code-Lösungen sind durch automatisierte Tests überprüfbar;
- Agenten können Lösungen iterieren, indem sie Testergebnisse als Feedback verwenden.
- Der Problembereich ist klar definiert und strukturiert; und
- Die Ausgabequalität kann objektiv gemessen werden.

In unserer eigenen Implementierung können Agenten nun echte GitHub-Probleme im [SWE-bench Verified Benchmark](#) allein anhand der [Pull-Request-Beschreibung](#) lösen. Während automatisierte Tests jedoch bei der Überprüfung der Funktionalität helfen, bleibt die menschliche Überprüfung entscheidend, um sicherzustellen, dass die Lösungen mit den breiteren Systemanforderungen übereinstimmen.

## Anhang 2: Schnelles Engineering Ihrer Tools

Unabhängig davon, welches agentische System Sie erstellen, werden Tools wahrscheinlich ein wichtiger Bestandteil Ihres Agenten sein. Tools ermöglichen es Claude, mit externen Diensten und APIs zu interagieren, indem sie deren genaue Struktur und Definition in unserer API festlegen. Wenn Claude antwortet, wird ein Tool-Use-Block in die API-Antwort aufgenommen, wenn ein Tool aufgerufen werden soll. Tool-Definitionen und -Spezifikationen sollten genauso schnell wie Ihren allgemeinen Eingabeaufforderungen Aufmerksamkeit geschenkt werden. In diesem kurzen Anhang beschreiben wir, wie Sie Ihre Tools zum Engineering auffordern.

Es gibt oft mehrere Möglichkeiten, dieselbe Aktion anzugeben. Sie können z. B. eine Dateibearbeitung angeben, indem Sie einen Diff schreiben oder die gesamte Datei neu schreiben. Für die strukturierte Ausgabe können Sie Code innerhalb von Markdown oder JSON zurückgeben. In der Softwareentwicklung sind solche Unterschiede kosmetischer Natur und können verlustfrei von einem in den anderen umgewandelt werden. Einige Formate sind für ein LLM jedoch viel schwieriger zu schreiben als andere. Um einen Diff zu schreiben, muss man wissen, wie viele Zeilen sich im Chunk-Header ändern, bevor der neue Code geschrieben wird. Das Schreiben von Code in JSON (im Vergleich zu Markdown) erfordert zusätzliches Escapen von Zeilenumbrüchen und Anführungszeichen.

Unsere Vorschläge für die Entscheidung für Werkzeugformate sind die folgenden:

- Geben Sie dem Modell genügend Token, damit es "denken" kann, bevor es sich selbst in eine Ecke schreibt.
- Halten Sie das Format nahe an dem, was das Modell natürlich in Text im Internet gesehen hat.
- Stellen Sie sicher, dass es keinen "Overhead" für die Formatierung gibt, z. B. die genaue Zählung von Tausenden von Codezeilen oder das String-Escapen von Code, der geschrieben wird.

Eine Faustregel lautet, darüber nachzudenken, wie viel Aufwand in Mensch-Computer-Schnittstellen (HCI) gesteckt wird, und ebenso viel Aufwand in die Erstellung guter Agent-Computer-Schnittstellen (ACI) zu investieren. Hier sind einige Gedanken, wie Sie dies tun können:

- Versetzen Sie sich in die Lage des Models. Ist es offensichtlich, wie Sie dieses Tool verwenden, basierend auf der Beschreibung und den Parametern, oder

müssen Sie sorgfältig darüber nachdenken? Wenn ja, dann gilt das wahrscheinlich auch für das Modell. Eine gute Werkzeugdefinition enthält häufig Beispiele für die Verwendung, Grenzfälle, Anforderungen an das EingabefORMAT und klare Grenzen zu anderen Werkzeugen.

- Wie können Sie Parameternamen oder Beschreibungen ändern, um die Dinge übersichtlicher zu machen? Stellen Sie sich dies so vor, als würden Sie einen großartigen Docstring für einen Junior-Entwickler in Ihrem Team schreiben. Dies ist besonders wichtig, wenn viele ähnliche Tools verwendet werden.
- Testen Sie, wie das Modell Ihre Werkzeuge verwendet: Führen Sie viele Beispieleingaben in unserem Arbeitsbereich aus, um zu sehen, welche Fehler das Modell macht, und iterieren Sie.
- Poka Yoke deine Werkzeuge. Ändern Sie die Argumente so, dass es schwieriger ist, Fehler zu machen.

Bei der Entwicklung unseres Agenten für SWE-bench haben wir tatsächlich mehr Zeit mit der Optimierung unserer Tools verbracht als mit der allgemeinen Eingabeaufforderung. Zum Beispiel haben wir festgestellt, dass das Modell Fehler mit Tools machte, die relative Dateipfade verwendeten, nachdem der Agent das Stammverzeichnis verlassen hatte. Um dies zu beheben, haben wir das Tool so geändert, dass immer absolute Dateipfade erforderlich sind – und wir haben festgestellt, dass das Modell diese Methode einwandfrei verwendet.



Claude	Presseanfragen	Nutzungsbedingung – Verbraucher
API	Unterstützen	Nutzungsbedingung – Kommerziell
Mannschaft	Status	Datenschutzrichtlin
Auszeichnung	Verfügbarkeit	Nutzungsbedingung
Forschung	Zwitschern	Richtlinie zur verantwortungsvolle Offenlegung
Codierung	LinkedIn (Englisch)	Beachtung
Firma	Auf YouTube (Englisch)	
Kundschaft		

Nachrichten

Karrieren

Datenschutz-  
Optionen

© 2025 Anthropisches PBC