Agentische KI-Muster und Workflows auf AWS

AWS Prescriptive Guidance

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

AWS Prescriptive Guidance: Agentische KI-Muster und Workflows auf AWS

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Die Handelsmarken und die Handelsaufmachung von Amazon dürfen nicht in einer Weise in Verbindung mit nicht von Amazon stammenden Produkten oder Services verwendet werden, durch die Kunden irregeführt werden könnten oder Amazon in schlechtem Licht dargestellt oder diskreditiert werden könnte. Alle anderen Handelsmarken, die nicht Eigentum von Amazon sind, gehören den jeweiligen Besitzern, die möglicherweise zu Amazon gehören oder nicht, mit Amazon verbunden sind oder von Amazon gesponsert werden.

Table of Contents

Einführung ... 1
Zielgruppe ... 1
Ziele ... 1
Über diese Inhaltsserie ... 2
Agentenmuster ... 3
Grundlagen der Argumentation ... 4
Architektur ... 4
Description ... 5
Capabilities ... 6
Einschränkungen ... 6
Häufige Anwendungsfälle ... 6
Implementierungsleitfaden ... 7
Zusammenfassung ... 7
Architektur ... 7
Description ... 8
Capabilities ... 9
Häufige Anwendungsfälle ... 9
Implementierungsleitfaden ... 9
Zusammenfassung ... 10
Tool-based Agenten zum Aufrufen von Funktionen ... 10
Architektur ... 10
Description ... 11
Capabilities ... 12
Häufige Anwendungsfälle ... 12
Implementierungsleitfaden ... 12
Zusammenfassung ... 13
Tool-based Agenten für Server ... 13
Architektur ... 13
Description ... 14
Capabilities ... 15
Häufige Anwendungsfälle ... 15
Implementierungsleitfaden ... 15
Zusammenfassung ... 16
Computer-use Agenten ... 16

Architektur ... 16
Description ... 17
Capabilities ... 18
Häufige Anwendungsfälle ... 18
Implementierungsleitfaden ... 19
Zusammenfassung ... 19
Codierungsagenten ... 19
Architektur ... 20
Description ... 20
Capabilities ... 21
Häufige Anwendungsfälle ... 21
Implementierungsleitfaden ... 22
Zusammenfassung ... 22
Sprach- und Sprachagenten ... 22
Architektur ... 22
Description ... 23
Capabilities ... 24
Häufige Anwendungsfälle ... 24
Implementierungsleitfaden ... 25
Zusammenfassung ... 25
Agenten zur Workflow-Orchestrierung ... 25
Architektur ... 26
Description ... 26
Capabilities ... 27
Häufige Anwendungsfälle ... 27
Implementierungsleitfaden ... 27
Zusammenfassung ... 28
Memory-augmented Agenten ... 28
Architektur ... 28
Description ... 29
Capabilities ... 30
Häufige Anwendungsfälle ... 30
Implementierung von Agenten mit erweitertem Speicher ... 30
Implementierung von Eingabeaufforderungen mit Speichereingabe ... 31
Zusammenfassung ... 32
Agenten für Simulation und Testumgebung ... 32

Architektur ... 32
Description ... 33
Capabilities ... 34
Häufige Anwendungsfälle ... 34
Implementierungsleitfaden ... 35
Zusammenfassung ... 36
Beobachter- und Überwachungsagenten ... 36
Architektur ... 37
Description ... 37
Capabilities ... 38
Häufige Anwendungsfälle ... 38
Implementierungsleitfaden ... 38
Zusammenfassung ... 39
Multi-agent Zusammenarbeit ... 40
Description ... 41
Capabilities ... 42
Häufige Anwendungsfälle ... 42
Implementierungsleitfaden ... 42
Zusammenfassung ... 43
Schlussfolgerung ... 44
Imbissbuden ... 44
LLM-Workflows ... 45
Überblick über Kognition LLM-augmented ... 45
Arbeitsablauf für schnelle Verkettung ... 46
Description ... 47
Capabilities ... 48
Häufige Anwendungsfälle ... 48
Arbeitsablauf für das Routing ... 48
Capabilities ... 50
Häufige Anwendungsfälle ... 50
Workflow für die Parallelisierung ... 50
Capabilities ... 51
Häufige Anwendungsfälle ... 51
Workflow für die Orchestrierung ... 52
Capabilities ... 53
Häufige Anwendungsfälle ... 53

Arbeitsablauf für Evaluatoren und Reflect-Refine-Schleifen ... 53
Häufige Anwendungsfälle ... 55
Capabilities ... 55
Schlussfolgerung ... 55
Agentäre Workflow-Muster ... 57
Von ereignisgesteuerten zu kognitionserweiterten Systemen ... 57
Event-driven Architektur ... 57
Cognition-augmented Arbeitsabläufe ... 58
Die wichtigsten Erkenntnisse ... 60
Sofortige Verkettung von Saga-Mustern ... 60
Saga-Choreographie ... 61
Muster für die schnelle Verkettung ... 62
Choreographie der Agenten ... 62
Imbissbuden ... 64
Weiterleitung dynamischer Versandmuster ... 64
Dynamischer Versand ... 65
LLM-based Routing ... 66
Agenten-Router ... 67
Imbissbuden ... 68
Parallelisierung und Scatter-Gather-Muster ... 68
Scatter-gather ... 69
LLM-based Parallelisierung (Scatter-Gather-Kognition) ... 71
Parallelisierung der Agenten ... 71
Imbissbuden ... 72
Orchestrierungsmuster von Saga ... 72
Orchestrierung von Veranstaltungen ... 73
Role-based Agentensystem (Orchestrator) ... 74
Supervisor ... 74
Imbissbuden ... 76
Schleifenmuster im Evaluator reflektieren/verfeinern ... 76
Regelkreis für Rückkopplung ... 77
Regelkreis für Rückkopplung (Evaluator) ... 79
Evaluator ... 79
Imbissbuden ... 80
Gestaltung agentischer Workflows auf AWS ... 81
Schlussfolgerung ... 81

Dokumentverlauf 83
..... lxxxiv

Agentische KI-Muster und Workflows auf AWS

Aaron Sempf und Andrew Hooker, Amazon Web Services

Juli 2025 (Geschichte der Dokumente)

Organizations setzen umfangreiche Sprachmodelle (LLMs) und Softwareagenten ein, um dynamische, domänenübergreifende Probleme mithilfe einer neuen Architekturdisziplin, den sogenannten Agentenmustern, zu lösen. Agentenmuster sind grundlegende Baupläne und modulare Konstrukte, die verwendet werden, um zielgerichtete KI-Agenten in vielen Kontexten zu entwerfen und zu orchestrieren.

Zielgruppe

Dieser Leitfaden richtet sich an Architekten, Entwickler und Produktleiter, die intelligente Anwendungen entwickeln möchten, die über statische Logik, symbolische Logik und deterministische Automatisierung hinausgehen.

Ziele

Dieser Leitfaden bietet einen Entwurfsrahmen und einen Implementierungsansatz für KI-Agentensysteme, die autonom arbeiten und gleichzeitig kontrollierbar bleiben und auf Ihre Ziele ausgerichtet sind. Es verbindet ereignisgesteuerte Architekturmuster mit verschiedenen agentischen Alternativen und zeigt, wie Sie mithilfe von Cloud-nativen Architekturen Agentensysteme für die Produktion aufbauen können. In diesem Leitfaden werden die folgenden Themen behandelt:

- Agentenmuster — Agentenmuster sind wiederverwendbare Entwurfsvorlagen, die die Struktur und das Verhalten einzelner Agenten beschreiben. Dazu gehören Agenten zur Argumentation, Agenten mit erweiterter Suchfunktion, Codierungsagenten, Sprachschnittstellen, Workflow-Orchestratoren und kollaborative Systeme mit mehreren Agenten. Jedes Muster veranschaulicht, wie Agenten wahrnehmen, argumentieren, handeln und lernen, und zwar anhand von Zuordnungen. AWS-Services
- LLM-Workflows — Workflows konzentrieren sich darauf, wie Agenten sie LLMs zum Argumentieren verwenden. Sie untersuchen Aufforderungsstrategien und Planungsmechanismen und skizzieren, wie LLMs sie nicht nur zur Textgenerierung, sondern auch zur Förderung strukturierter, interpretierbarer und zuverlässiger Verhaltensweisen innerhalb einer Agentenschleife eingesetzt werden.

- Agentische Workflow-Muster — Workflow-Muster beschreiben, wie mehrere Agenten, Tools und Umgebungen interagieren, um autonome Systeme zu bilden. Dazu gehören Muster für die Orchestrierung von Aufgaben, die Delegierung von Subagenten, die ereignisbasierte Koordination, Beobachtbarkeit und Kontrolle. Diese Aspekte fördern skalierbare, zusammensetzbare und überprüfbare KI-Architekturen.

Über diese Inhaltsserie

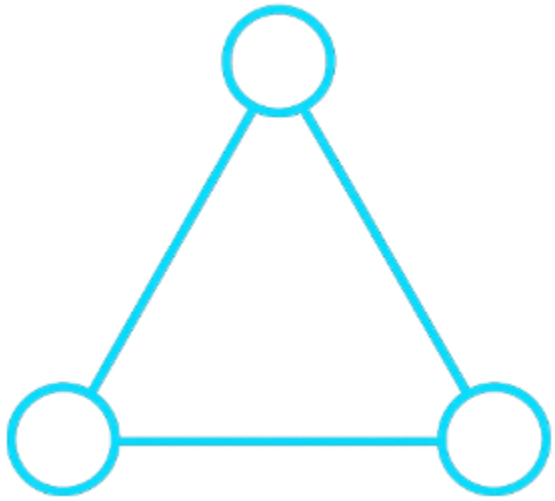
Dieser Leitfaden ist Teil einer Reihe über agentic AI on. AWS Weitere Informationen und die anderen Leitfäden dieser Reihe finden Sie unter Agentic AI auf der Prescriptive Guidance-Website. AWS

Agentenmuster

Agentenmuster sind wiederverwendbare, zusammensetzbare Bausteine, die auf bestimmte Domänen, Anwendungsfälle und Komplexitätsstufen zugeschnitten werden können. Agentensysteme unterscheiden sich jedoch von herkömmlichen Anwendungen. Im Mittelpunkt aller Designs von KI-Agenten steht ein konzeptionelles Modell, das in den folgenden drei Grundprinzipien verankert ist:

- Asynchron — Agenten arbeiten in lose gekoppelten, ereignisreichen Umgebungen
- Autonomie — Agenten agieren unabhängig, ohne menschliche oder externe Kontrolle
- Entscheidungsfreiheit — Agenten handeln zielgerichtet, im Namen eines Benutzers oder Systems, um bestimmte Ziele zu erreichen

Das Dreieck in der folgenden Abbildung stellt die Kernbausteine eines Softwareagenten dar: Wahrnehmung, Vernunft und Handlung. Dies ermöglicht es einem Agentensystem, seine Umgebung zu beobachten, Entscheidungen zu treffen und in ihr zu agieren.



Agentenmuster bieten von Natur aus eine modulare Designsprache für den Aufbau von KI-Systemen, was bedeutet, dass sie zugänglich, betriebsbereit, erweiterbar und produktionsbereit sind. Bei der Entwicklung dieser Systeme müssen die folgenden drei miteinander verbundenen Dimensionen sorgfältig beachtet werden, auf die weiter unten in diesem Leitfaden näher eingegangen wird.

In diesem Abschnitt

- Grundlagen der Argumentation
- Tool-based Agenten zum Aufrufen von Funktionen

- Tool-based Agenten für Server
- Computer-use Agenten
- Codierungsagenten
- Sprach- und Sprachagenten
- Agenten zur Workflow-Orchestrierung
- Memory-augmented Agenten
- Agenten für Simulation und Testumgebung
- Beobachter- und Überwachungsagenten
- Multi-agent Zusammenarbeit

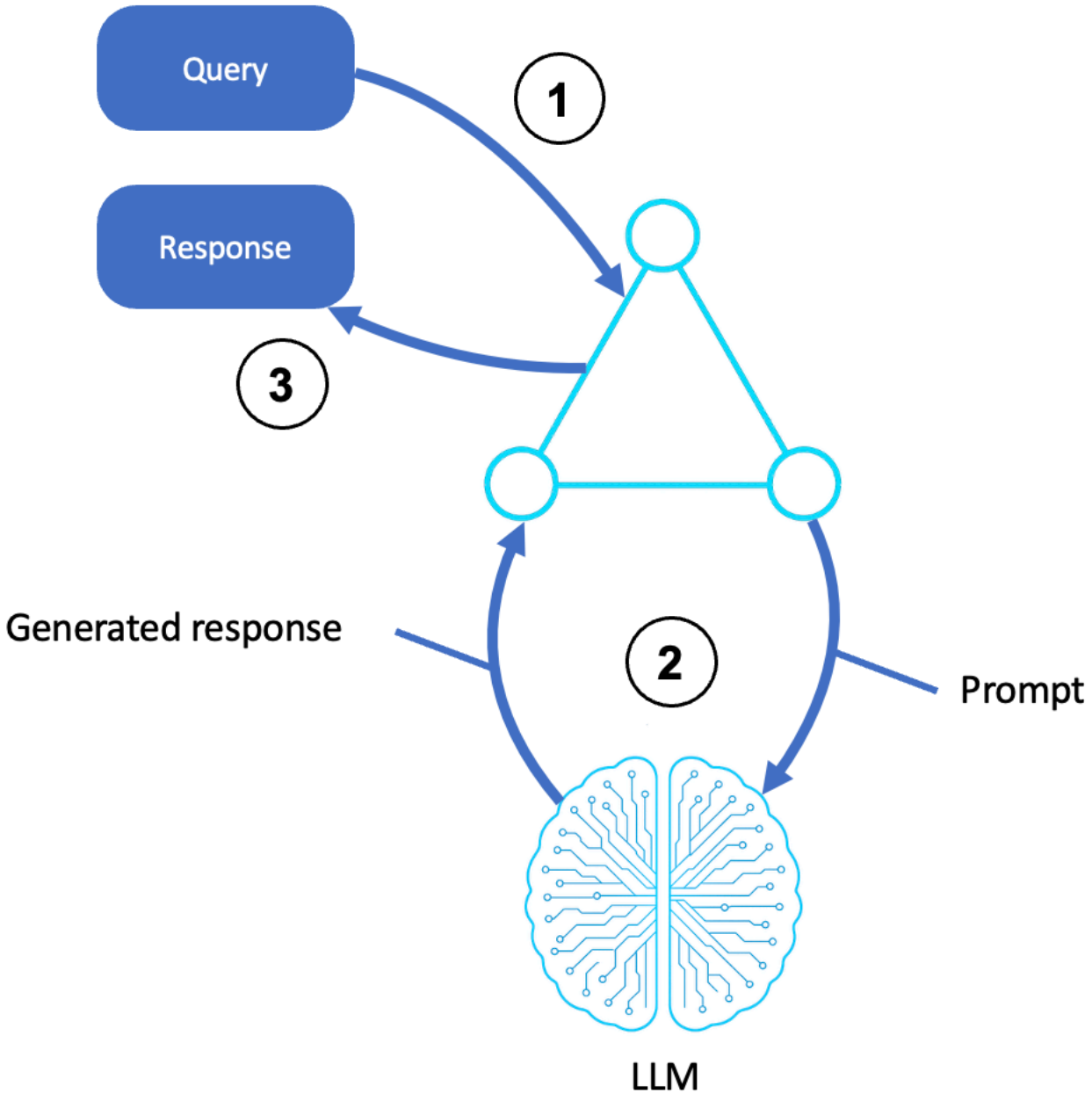
Grundlagen der Argumentation

Ein Agent für grundlegendes Denken ist die einfachste Form der agentischen KI, die als Antwort auf eine Anfrage logische Folgerungen oder Entscheidungen trifft. Es akzeptiert Eingaben von Benutzern oder Systemen und verarbeitet Abfragen und generiert Antworten mithilfe strukturierter Eingabeaufforderungen.

Dieses Muster ist nützlich für Aufgaben, die eine einstufige Argumentation, Klassifizierung oder Zusammenfassung auf der Grundlage eines bestimmten Kontextes erfordern. Es verwendet keinen Arbeitsspeicher, keine Tools oder Statusverwaltung, wodurch es zustandslos, leichtgewichtig und in großen Workflows gut zusammensetzbar ist.

Architektur

In der folgenden Abbildung wird der Ablauf eines Argumentationsmittels dargestellt:



Description

1. Empfängt eine Eingabe
   - Ein Benutzer, ein System oder ein Upstream-Agent sendet eine Anfrage oder Anweisung.
   - Die Eingabe wird an die Agenten-Shell oder die Orchestrierungsebene übergeben.
   - Dieser Schritt umfasst die Vorverarbeitung, die Erstellung von Vorlagen für Eingabeaufforderungen und die Identifizierung der Ziele.

2. Ruft das LLM auf
   - Der Agent wandelt die Anfrage in eine strukturierte Aufforderung um und sendet sie an ein LLM (z. B. über Amazon Bedrock).
   - Das LLM generiert auf der Grundlage der Aufforderung eine Antwort, wobei vorab geschultes Wissen und Kontext verwendet werden.
   - Das generierte Ergebnis kann Argumentationsschritte (Gedankenkette), endgültige Antworten oder Ranglisten beinhalten.
3. Gibt eine Antwort zurück
   - Die generierte Ausgabe wird an die Schnittstelle des Agenten weitergeleitet.
   - Dies kann Formatierung, Nachbearbeitung oder eine API-Antwort beinhalten.

Capabilities

- Unterstützt natürliche Sprache oder strukturierte Eingabe
- Nutzt schnelle Technik, um das Verhalten zu steuern
- Zustandslos und skalierbar
- Kann in UI, CLI, APIs und Pipelines eingebettet werden

Einschränkungen

- Kein Gedächtnis oder Geschichtsbewusstsein
- Keine Interaktion mit externen Tools oder Datenquellen
- Beschränkt auf das, was der LLM zum Zeitpunkt der Schlussfolgerung weiß

Häufige Anwendungsfälle

- Konversationsfragen und Antworten
- Erläuterungen und Zusammenfassungen der Richtlinien
- Leitlinien für die Entscheidungsfindung
- Leichte und automatisierte Chatbot-Abläufe
- Klassifizierung, Kennzeichnung und Bewertung

Implementierungsleitfaden

Sie können die folgenden Tools und Dienste verwenden, um ein grundlegendes Argumentationsinstrument zu erstellen:

- Amazon Bedrock für LLM-Aufrufe (Anthropic, AI21, Meta)
- Amazon API Gateway oder AWS Lambda um es als zustandslosen Microservice verfügbar zu machen
- Rufen Sie Vorlagen auf, die im Parameter Store, AWS Secrets Manager, oder als Code gespeichert sind

Zusammenfassung

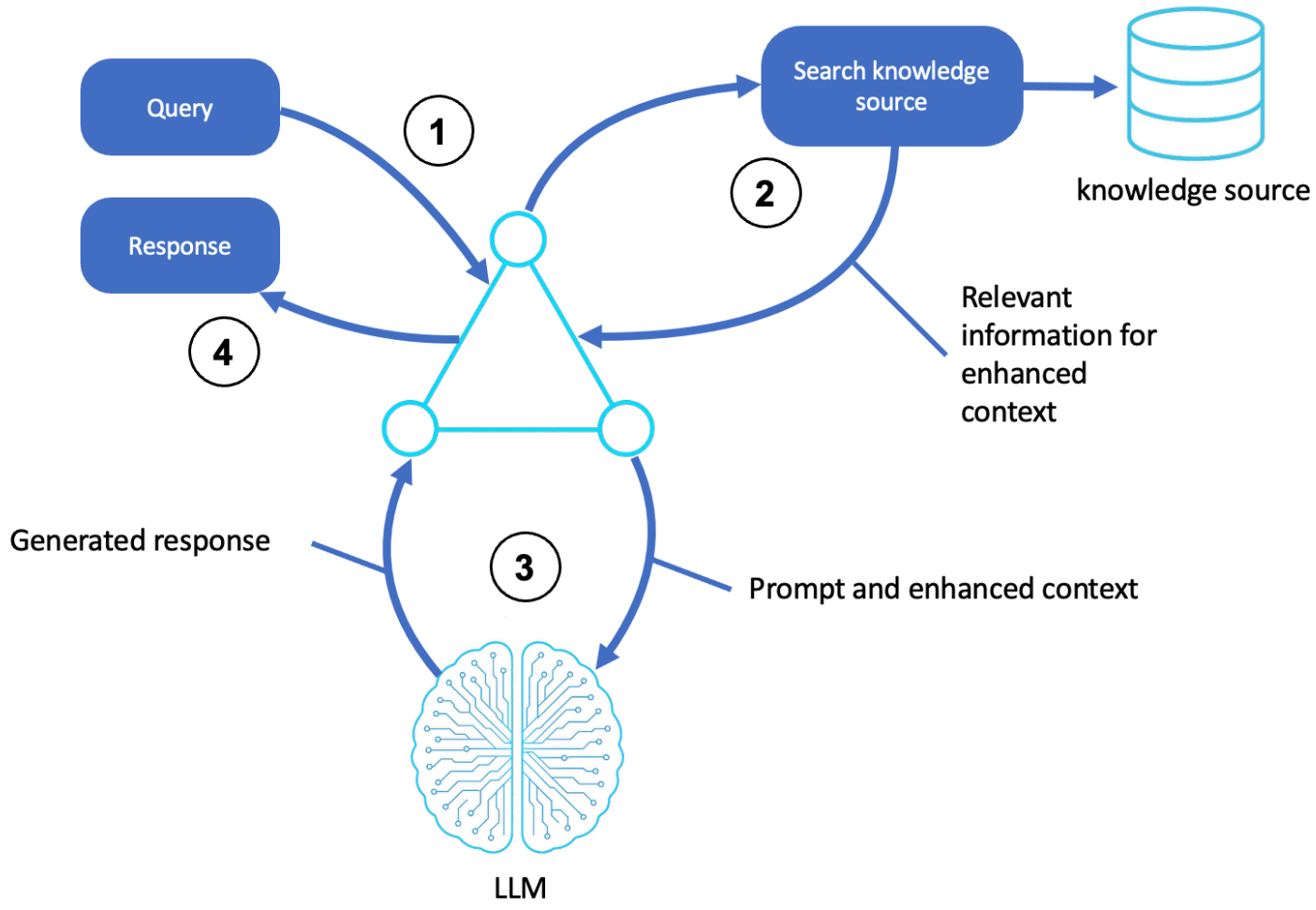
Das grundlegende Argumentationsmittel ist aufgrund seiner einfachen Struktur von grundlegender Bedeutung. Es verfügt über Kernfunktionen, die Ziele in Argumentationswege umwandeln, die zu intelligenten Ergebnissen führen. Dieses Muster ist häufig der Ausgangspunkt für fortgeschrittene Muster, wie z. B. toolbasierte Agenten und Agenten, die Retrieval-Augmented Generation (RAG) verwenden. Es ist auch eine zuverlässige und modulare Komponente großer Workflows.

Agentin RAG

Retrieval-augmented Generierung (RAG) ist eine Technik, die das Abrufen von Informationen mit der Textgenerierung kombiniert, um genaue und kontextbezogene Antworten zu erhalten. RAG ermöglicht es Agenten, relevante externe Informationen abzurufen, bevor sie das LLM in Anspruch nehmen. Es erweitert das effektive Gedächtnis und die Argumentationsgenauigkeit eines Agenten, indem es seine Entscheidungen auf aktuelle, sachliche oder domänenspezifische Informationen stützt. Im Gegensatz zu staatenlosen LLMs, die sich ausschließlich auf vortrainierte Gewichte verlassen, verfügt RAG über eine externe Wissenssuchebene, die Eingabeaufforderungen dynamisch an den Kontext anpasst.

Architektur

Die Logik des RAG-Musters wird in der folgenden Abbildung veranschaulicht:



Description

1. Empfängt eine Anfrage
   - Ein Benutzer oder ein Upstream-System sendet eine Anfrage oder ein Ziel an den Agenten.
   - Die Agenten-Shell akzeptiert die Anfrage und formatiert sie als Aufforderung zur Begründung.
2. Durchsucht eine externe Quelle
   - Der Agent identifiziert Konzepte und Absichten anhand der Abfrage.
   - Er fragt mithilfe der semantischen Suche oder des Stichwortabgleichs eine Wissensquelle ab, z. B. einen Vektorspeicher, eine Datenbank oder einen Dokumentenindex.
   - Die relevantesten Passagen, Dokumente oder Entitäten werden zur Verwendung im nächsten Schritt abgerufen.
3. Generiert eine kontextuelle Antwort

- Der Agent erweitert die Eingabeaufforderung um die abgerufenen Informationen und bildet so eine kontexterweiterte Eingabe für das LLM.
- Das LLM verarbeitet alle Eingaben mithilfe von generativem Denken (z. B. Gedankenkette oder Reflexion), um eine genaue Antwort zu erhalten.

4. Gibt die endgültige Ausgabe zurück
   - Der Agent bereitet die Ausgabe vor, indem er sie in alle Kommunikationskopfzeilen oder erforderlichen Formatierungen verpackt und sie dann an den Benutzer oder das aufrufende System zurückgibt.
   - (Optional) Die abgerufenen Dokumente und die LLM-Ausgabe können protokolliert, bewertet und für future Abfragen im Speicher gespeichert werden.

Capabilities

- Fact-grounded Ausgabe auch in Long-Tail- oder unternehmensspezifischen Domänen
- Speichererweiterung ohne Feinabstimmung des Modells
- Dynamischer Kontext, der auf jeder Abfrage und jedem Benutzerstatus basiert
- Vollständig kompatibel mit Vektordatenbanken, semantischen Indizes und Metadatenfilterung

Häufige Anwendungsfälle

- Wissensassistenten für Unternehmen
- Bots zur Einhaltung gesetzlicher Vorschriften
- Copiloten für den Kundensupport
- Search-enhanced Chatbots
- Agenten für die Dokumentation für Entwickler

Implementierungsleitfaden

Verwenden Sie die folgenden Tools und Dienste, um einen Agenten zu erstellen, der RAG verwendet:

- Amazon Bedrock für LLM-Aufruf

- Amazon Kendra oder Amazon Aurora für Dokumentation oder strukturierte Datensuche OpenSearch
- Amazon Simple Storage Service (Amazon S3) für die Aufbewahrung von Dokumenten
- AWS Lambda um Suche, Eingabeaufforderung und LLM-Inferenz zu orchestrieren
- Knowledge-based Integrationen mit Agenten (mithilfe von Speicher-Plugins, semantischen Retrievern oder Amazon Bedrock)

Zusammenfassung

Agent RAG verbindet statisches Modelldenken mit dynamischer Intelligenz aus der realen Welt. Es gibt Agenten die Möglichkeit, nachzuschlagen, was sie nicht wissen, Antworten aus abgerufenem Wissen zu synthetisieren und zuverlässige, überprüfbare Antworten zu erstellen.

RAG-Muster sind eine Grundlage für die Entwicklung intelligenter Agenten, die den Wissenszugang ohne Umschulung skalieren können. Es ist oft ein Vorläufer für komplexere Orchestrierungsmuster, die den Einsatz von Tools, die Planung und das Langzeitgedächtnis beinhalten.

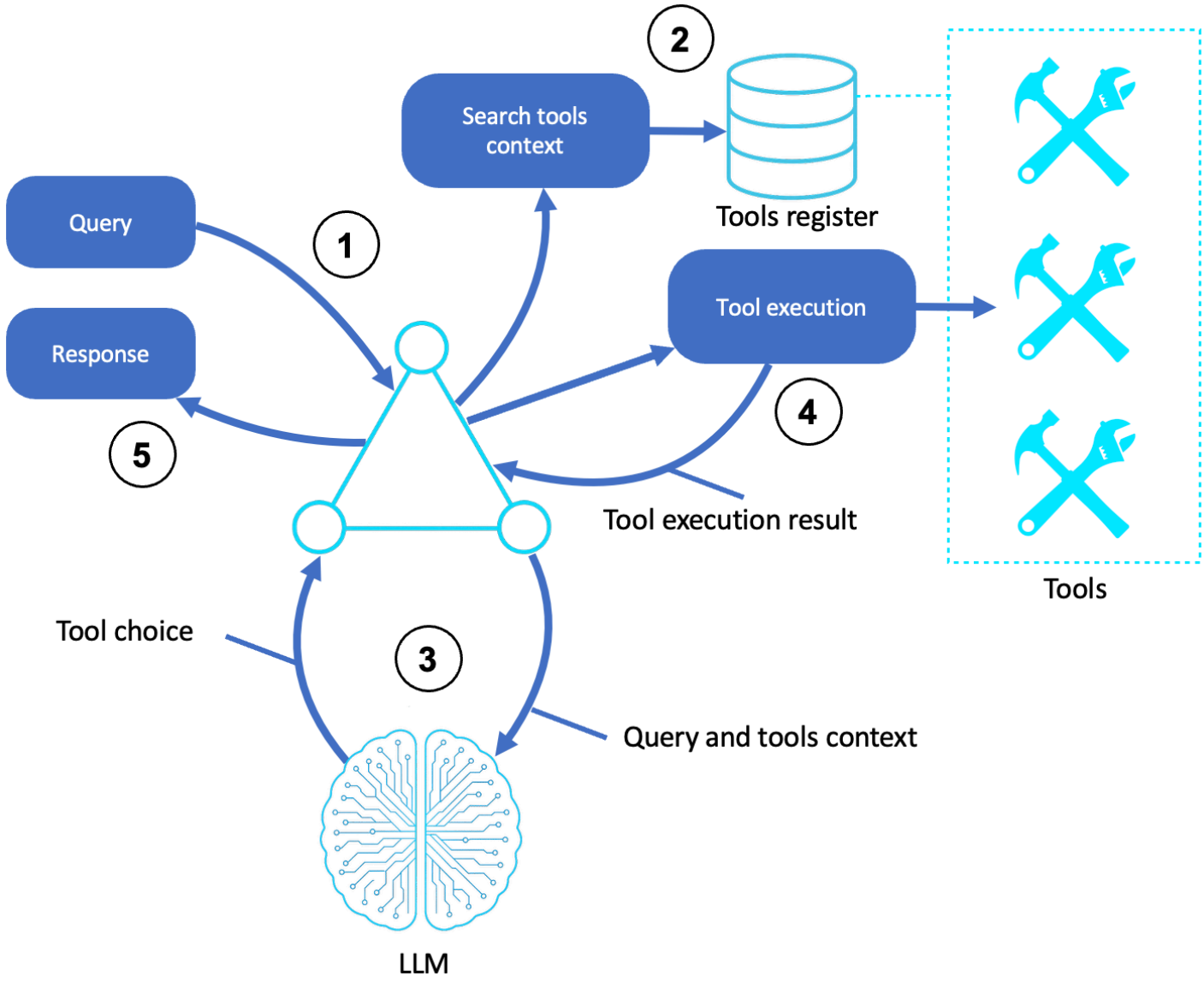
Tool-based Agenten zum Aufrufen von Funktionen

Tool-based Agenten erweitern die Fähigkeiten von Argumentationsagenten, indem sie externe Funktionen oder APIs aufrufen, um Aufgaben zu erledigen, die über das rein sprachliche Denken hinausgehen. Dieses Muster verwendet ein LLM, um zu entscheiden, welches Tool verwendet werden soll, generiert dann Anrufargumente und bezieht die Ergebnisse eines Tools in die Argumentationsschleife ein.

Dieses Muster ermöglicht es den Agenten, zu handeln, anstatt nur Antworten zu geben. Die Tool-Schnittstelle steht für jede aufrufbare Funktion, die von arithmetischen Berechnungen und Datenbanksuchen bis hin zu externen APIs und Cloud-Diensten reicht.

Architektur

In der folgenden Abbildung ist ein toolbasierter Agent zum Aufrufen von Funktionen dargestellt:



Description

1. Empfängt eine Anfrage
   - Der Agent erhält eine Anfrage oder Aufgabe in natürlicher Sprache vom Benutzer oder vom aufrufenden System.
2. Sucht nach Tools
   - Der Agent verwendet interne Metadaten oder eine Tool-Registry, um nach verfügbaren Tools, Schemas und relevanten Funktionen zu suchen.
3. Wählt Tools aus und ruft sie auf

- Das LLM empfängt die Abfrage- und Tool-Metadaten (z. B. Funktionsnamen, Eingabetypen und Beschreibungen) in seiner Eingabeaufforderung.
- Es wählt das relevanteste Tool aus, konstruiert Eingabeargumente und gibt einen strukturierten Funktionsaufruf zurück.

4. Führt das gewählte Tool aus
   - Die Agent-Shell oder der Tool Runner führt die ausgewählte Funktion aus und gibt das Ergebnis zurück (z. B. eine API-Ausgabe, einen Datenbankwert oder eine Berechnung).
5. Gibt eine Antwort zurück
   - Das LLM leitet die Ergebnisse entweder direkt oder als Teil einer aktualisierten Aufforderung an den Agenten weiter. Anschließend wird ein Ergebnis in natürlicher Sprache zurückgegeben.

Capabilities

- Dynamische Werkzeugauswahl basierend auf dem Aufgabenkontext
- Schema-based Aufforderung (OpenAPI, JSON-Schema, AWS Funktionsschnittstelle)
- Interpretation der Ergebnisse und Verkettung der Ergebnisse zu Argumenten
- Zustandslose oder sitzungsabhängige Operationen

Häufige Anwendungsfälle

- Virtuelle Assistenten mit externem Datenzugriff
- Finanzrechner und Schätzer
- API-based Wissensarbeiter
- LLMs, die SageMaker Amazon-Endpunkte AWS Lambda und SaaS-Dienste aufrufen

Implementierungsleitfaden

Verwenden Sie Folgendes, um toolbasierte Agenten zum Aufrufen von Funktionen zu erstellen:

- Amazon Bedrock mit Unterstützung für Funktionsaufrufe (Anthropic Claude)
- AWS Lambda als Backend für die Ausführung von Tools
- Amazon API Gateway oder AWS Step Functions für die Orchestrierung von Tools

- Amazon DynamoDB oder Amazon Relational Database Service (Amazon RDS) für kontextsensitive Tool-Metadaten
- EventBridge Amazon-Pipelines oder AWS Step Functions die Staaten zuordnen, um Ausgaben weiterzuleiten

Zusammenfassung

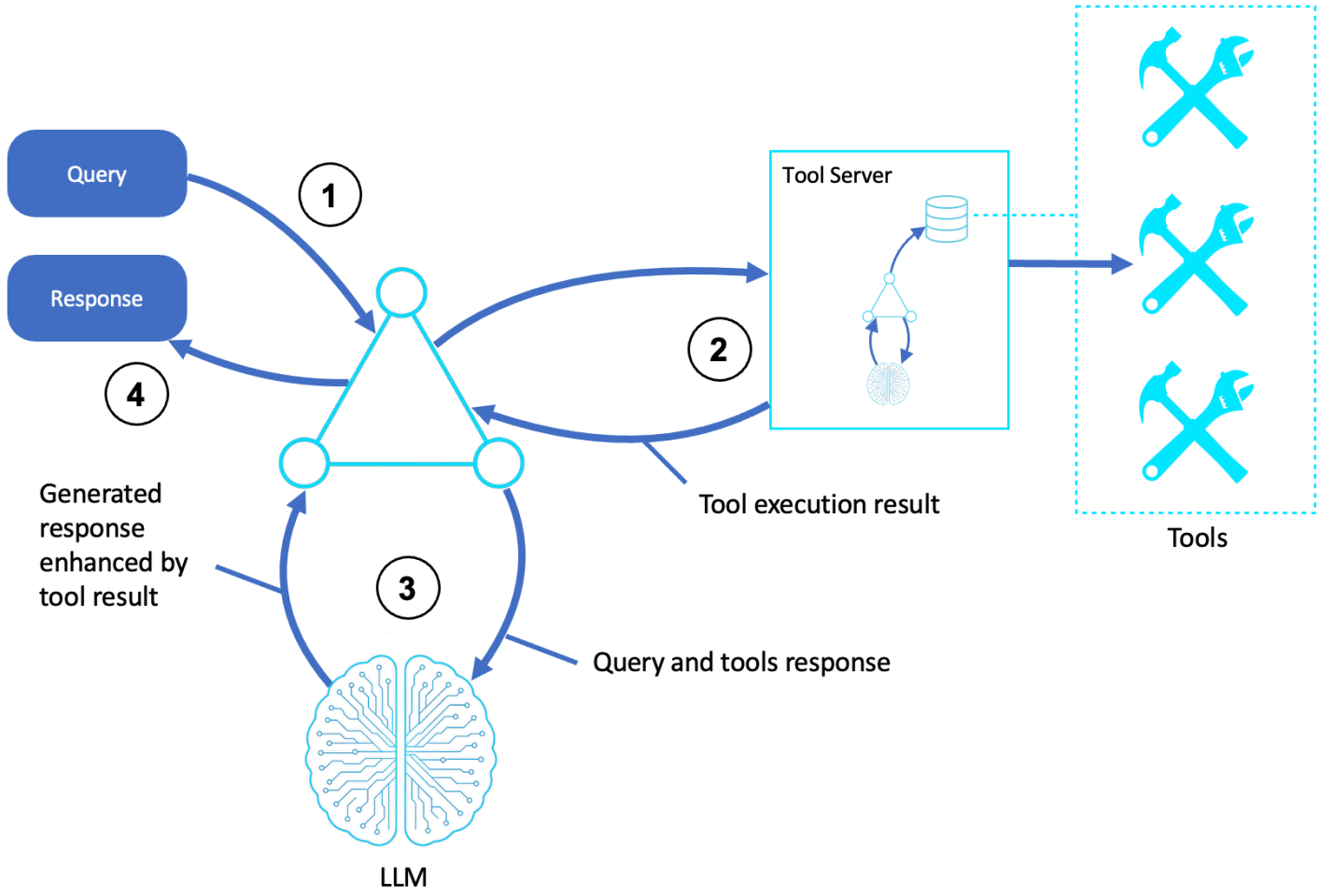
Tool-based Agenten, die Funktionen aufrufen, bedeuten eine Verlagerung vom Sprachverständnis hin zur Ausführung von Aktionen. Diese Agenten nutzen dynamische, kontextsensitive Tools und behalten gleichzeitig die LLM-Argumentation bei. Sie verwandeln passive Assistenten in Systeme, die Aufgaben erledigen, auf Dienste zugreifen und Geschäftsabläufe integrieren. Dieses Muster ist ein wichtiger Bestandteil der agentischen KI in Unternehmensumgebungen, insbesondere in Kombination mit deklarativen Schemas, Autorisierungs-Frameworks und Systemen mit mehreren Agenten.

Tool-based Agenten für Server

Tool-based Agenten für Server verbessern die Funktion von Agenten, die Funktionen aufrufen, indem sie die Ausführung von Tools an einen externen Server delegieren, der über eine spezielle Laufzeitumgebung für Tools, Skripts und Verbundagenten verfügt. Im Gegensatz zu Inline-Funktionsaufrufen, die in der Agentenschleife ausgewählt und aufgerufen werden, lagern serverbasierte Agenten die Logik und die Ausführungspipeline an andere Agenten oder Systeme aus. Dies bietet erweiterte Funktionen wie Multitool-Verkettung, isolierte Ausführung und spezielle Argumentation. Toolserver eignen sich ideal für komplexe, zustandsbehaftete oder ressourcenintensive Aktionen, bei denen die Tools selbst separate KI-Modelle, Geschäftsregeln oder Umgebungen beinhalten können.

Architektur

Im Folgenden finden Sie ein Muster für toolbasierte Agenten für Server:



Description

1. Empfängt eine Anfrage
   - Ein Benutzer oder ein System sendet eine Anfrage an die Agenten-Shell.
   - Der Agent interpretiert die Abfrage und bereitet ihren Versand an einen Toolserver vor.
2. Führt Toolserver-Prozesse aus
   - Der Agent sendet die Aufgabe zusammen mit strukturierten Parametern an einen Toolserver.
   - Der Tool-Server kann dann:
     - Führen Sie Skripts oder Logik in speziellen Rechensystemen aus (AWS Lambda z. B. Container oder Amazon SageMaker)
     - Verwenden Sie einen eigenen Subagenten mit LLM-Argumentation, um Tools auszuwählen und auszuführen
     - Verwalten Sie Abhängigkeiten, Wiederholungsversuche oder mehrstufige Ausführungsabläufe

    - Gibt die Ergebnisse an den primären Agenten aus, wenn die Aufgabe abgeschlossen ist
3. Verwendet LLM-Argumentation bei der Toolausgabe
    - Der Agent ruft ein LLM auf und übergibt die ursprüngliche Abfrage und das Ergebnis des Toolservers als Teil der Eingabeaufforderung.
    - Das LLM synthetisiert eine Antwort, die die neu erfassten Informationen enthält.
4. Gibt eine Antwort zurück
    - Der Agent gibt eine Antwort in natürlicher Sprache oder in strukturierter Form an den Benutzer oder das anrufende System zurück.
    - (Optional) Die Ergebnisse können im Speicher oder in Auditprotokollen gespeichert werden.

Capabilities

- Tools werden außerhalb der Ausführungsschleife des primären Agenten aufgerufen
- Die Ausführung von Tools kann LLM-Aufrufe, Logikketten oder Subagenten beinhalten
- Der Agent fungiert als Controller oder Dispatcher, nicht nur als Tool-Wrapper
- Ermöglicht Zusammensetzbarkeit, Skalierbarkeit und Isolierung der Logik

Häufige Anwendungsfälle

- Orchestrierung von Modellketten (z. B. durch Kombination von LLM, Vision und Code)
- AI-driven Automatisierungspipelines
- DevOps Assistenzagenten mit Script-Runnern
- Agenten für komplexe Finanzberechnungen, Simulationen oder Optimierungen
- Multimodale Tools (z. B. durch die Kombination von Audio, Dokumentation und Aktion)

Implementierungsleitfaden

Sie können dieses Muster wie folgt AWS-Services erstellen:

- Amazon Bedrock (Agentenhost und LLM-Inferenz)
- AWS Lambda, Amazon ECS oder SageMaker Amazon-Endpoints als Tool-Server-Runtime AWS Fargate
- Amazon API Gateway oder AWS App Runner um Toolserver-APIs verfügbar zu machen

- Amazon EventBridge für entkoppeltes Agent-zu-Tool-Messaging
- AWS Step Functions oder AWS AppFabric für die Erstellung von Multi-Agent-Logik auf dem Tool-Server

Zusammenfassung

Tool-based Agenten, die Server verwenden, sind hochgradig modular und skalierbar. Sie entkoppeln die Entscheidungslogik von der Ausführung, was es dem primären Agenten ermöglicht, schlank zu bleiben und gleichzeitig komplexe oder sensible Aktionen auf andere Systeme auszulagern. Dies ist wichtig für agentische KI auf Unternehmensebene, insbesondere in Umgebungen, in denen Steuerung, Beobachtbarkeit, Isolierung, dynamische Zusammensetzung oder eine beliebige Kombination davon erforderlich sind.

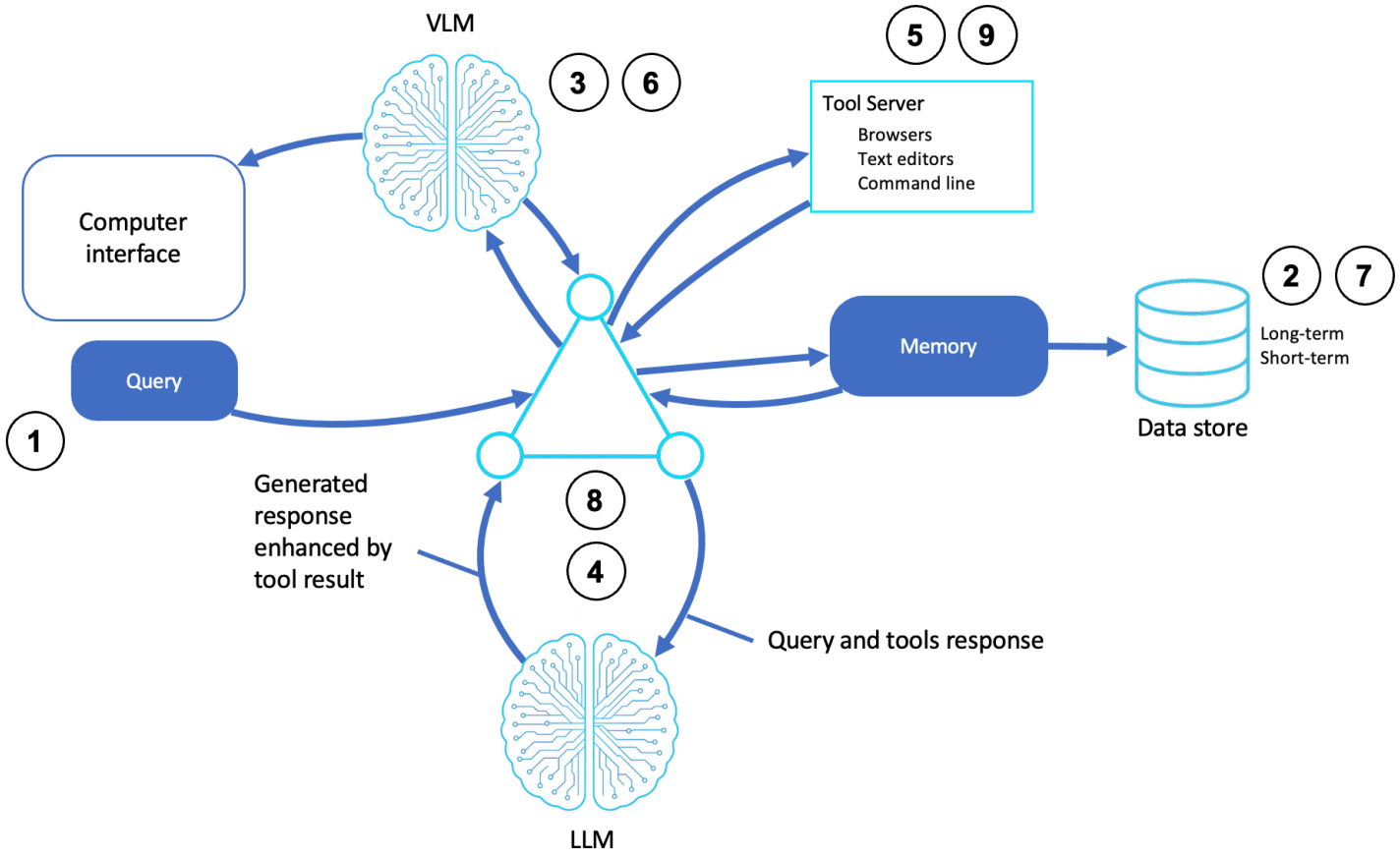
Computer-use Agenten

Computer-use Agenten können digitale Umgebungen wie Browser, Terminals, Dateisysteme und Anwendungen simulieren oder steuern. Diese Agenten interpretieren Benutzerabsichten, interagieren mit visuellen und textlichen Benutzeroberflächen und führen zielgerichtete Aktionen durch, indem sie LLM-Argumentation, visuelle Sprachmodelle (VLMs) und Toolserver kombinieren, die Befehle ausführen oder Eingabeereignisse simulieren.

Dieses Muster ist wichtig für praktische KI-Automatisierungen, bei denen der Agent nicht nur als Assistent, sondern auch als Proxy fungiert, der Aktionen wie ein Mensch ausführt, häufig unter Verwendung derselben Tools und Umgebungen.

Architektur

In der folgenden Abbildung ist ein Muster für die Computernutzung durch Agenten dargestellt:



Description

1. Empfängt eine Anfrage
   - Eine Aufgabe oder Anfrage wird über eine Benutzeroberfläche, eine API oder eine Schnittstelle in natürlicher Sprache bereitgestellt.
2. Greift auf den Speicher zu
   - Der Agent ruft das Kurz- und Langzeitgedächtnis ab, um vergangene Befehle, Ziele und Systemzustände abzurufen.
3. Analysiert den visuellen Kontext
   - Ein VLM beobachtet den Computerbildschirm, den Systemstatus oder Elemente der Benutzeroberfläche, um einen bestimmten Kontext zu verstehen und umsetzbare Elemente zu identifizieren.
4. Gründe anhand eines LLM
   - Das LLM kombiniert die Abfrage, den Speicherstatus, das Tool und die Serverantwort, um die nächste Aktion zu bestimmen.

5. Interagiert mit dem Tool-Server
   - Der Agent ruft Tools auf, die auf einem Server gehostet werden. Dazu können folgende Tools gehören:
     - Browser (z. B. Headless Chrome) und Shell-Umgebungen
     - Text- und Code-Editoren
     - Benutzerdefinierte Skriptschnittstellen
6. Aktualisiert visuelle Eingaben
   - Wenn sich die Benutzeroberfläche des Systems ändert oder weitere Beobachtungen erforderlich sind, analysiert das VLM möglicherweise den Bildschirmstatus oder die Textpuffer erneut.
7. Aktualisiert den Speicher
   - Neue Erkenntnisse, Systemzustände oder Benutzerfeedback werden in das Kurz- und Langzeitgedächtnis geschrieben.
8. Formuliert endgültige Entscheidungen und Erklärungen
   - Das LLM fasst die Ergebnisse zusammen oder empfiehlt Maßnahmen auf der Grundlage der Abfrage und der Toolausgabe.
9. Gibt eine Antwort zurück
   - Der Agent gibt Ergebnisse an die Schnittstelle zurück (z. B. eine abgeschlossene Aufgabe, eine Bestätigung oder generierter Inhalt).

Capabilities

- Multimodales Denken mit visuellen und textuellen Eingaben
- Kontrolle über Anwendungen durch simulierte oder Eingaben API-driven
- Speicherverwaltung für persistenten Zustand
- Autonomie bei der Sequenzausführung (mehrstufige Abläufe)

Häufige Anwendungsfälle

- KI-Entwickler, die Code in IDEs schreiben und ausführen
- Computer-use Agenten für sich wiederholende digitale Workflows
- Simulierte Benutzer für Softwaretests und Qualitätssicherung

- Agenten für Barrierefreiheit zum Navigieren in Benutzeroberflächen mithilfe von Sprachanweisungen oder Anweisungen auf hoher Ebene
- Intelligente robotergestützte Prozessautomatisierung (RPA), die durch logisches Denken erweitert wird

Implementierungsleitfaden

- Sie können dieses Muster wie folgt erstellen: AWS-Services
- Amazon Bedrock für LLM-based Planung und Argumentation
- Amazon Elastic Compute Cloud (Amazon EC2) oder SageMaker Amazon-Notebooks zum Ausführen von Toolservern mit simulierten Benutzeroberflächenumgebungen AWS Lambda
- Amazon Simple Storage Service (Amazon S3) oder Amazon DynamoDB für Speicherpersistenz
- Amazon Rekognition (oder benutzerdefinierte Modelle) für die UI-Bildanalyse in hybriden Szenarien
- Amazon CloudWatch Logs oder AWS X-Ray für Beobachtbarkeit und Prüfprotokolle

Zusammenfassung

Computer-use Agenten agieren als autonome digitale Operatoren und überbrücken die Lücke zwischen Interaktionen und Aktionen zwischen Mensch und Computer. AI-driven Durch die Integration von Speicher, Werkzeugorchestrierung und VLMs können diese Agenten adaptiv mit Systemen interagieren, die für Menschen konzipiert sind, Aktionen ausführen, Dateien aktualisieren, in Menüs navigieren und Antworten generieren.

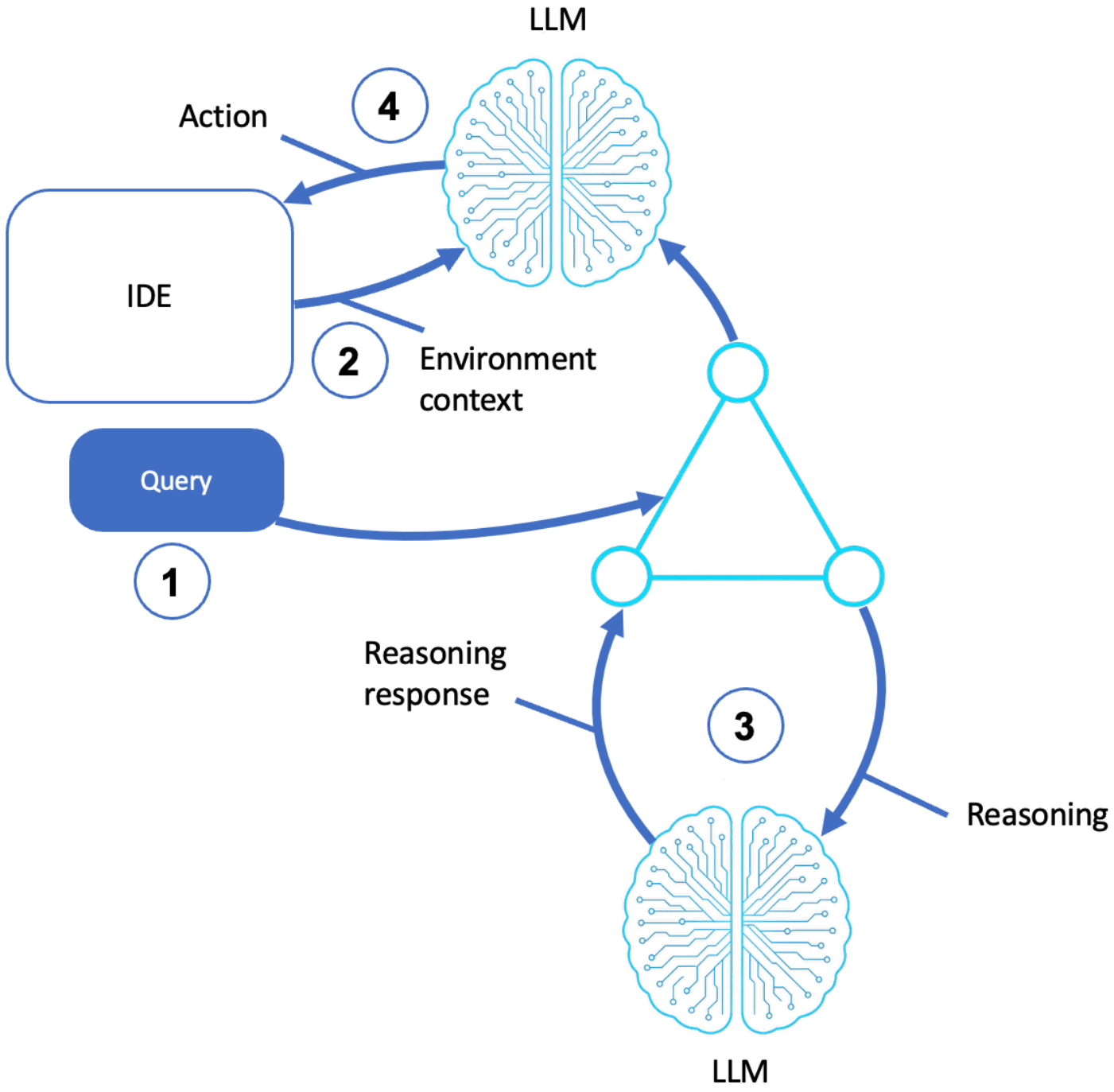
Codierungsagenten

Programmieragenten können über Programmieraufgaben nachdenken, Code generieren oder ändern und mit Entwicklerumgebungen wie IDEs und CLIs interagieren. Diese Agenten kombinieren natürliches Sprachverständnis mit strukturiertem Denken, um die Softwareentwicklung zu unterstützen, zu erweitern und zu automatisieren — von der Funktionsgenerierung über die Fehlerbehebung bis hin zur Erstellung von Tests.

Im Gegensatz zu Tools zur automatischen Vervollständigung interpretieren Programmieragenten aktiv die Ziele der Benutzer, fragen die Entwicklungsumgebung nach Kontext ab (sie öffnet beispielsweise Dateien und verfolgt Fehler), identifizieren Anforderungen und schlagen dann Aktionen vor und führen sie durch.

Architektur

Das folgende Diagramm zeigt ein Muster für Codierungsagenten:



Description

1. Empfängt eine Anfrage

- Der Benutzer stellt Anweisungen in natürlicher Sprache über eine Befehlspalette, ein Chat-Fenster oder eine CLI bereit (z. B. „Protokollierung zu dieser Funktion hinzufügen“ oder „Aus Gründen der Lesbarkeit umgestalten“).

2. Extrahiert den Umgebungskontext
   - Der Agent sammelt den Kontext aus der IDE, einschließlich aktiver Dateien, Cursorposition, Codefragmente und Symboltabellen.
   - Er gibt Fehlermeldungen, Testergebnisse und Ausgaben von anderen Agenten aus.
3. LLM-Argumentation
   - Der Agent sendet eine Aufforderung, einschließlich der Anfrage und des Umgebungskontextes, an einen LLM.
     - Das LLM führt einen Argumentationsprozess durch, um Folgendes festzustellen:
     - Was muss sich ändern
     - Wie generiert man eine Lösung
     - Alle Schritte zum Refactoring, Umschreiben oder Codieren
4. Führt Aktionen aus
   - Das LLM gibt die Ausgabe an den Agenten zurück und importiert sie in die IDE- oder Laufzeitumgebung.
   - Dies kann das Einfügen oder Ändern von Code, das Generieren von Kommentaren oder Dokumentation und das Auslösen nachgelagerter Build-, Test- und Lint-Aufgaben beinhalten.

Capabilities

- High-contextual Sensibilisierung (z. B. IDE-Status, Cursor und Syntaxbaum)
- Iterative Argumentation von Zielen und Feedback
- Optionale Codeplanung und Trennung von Aktionen (z. B. zuerst begründen und dann handeln)
- Funktioniert in synchronen oder asynchronen Entwickler-Workflows

Häufige Anwendungsfälle

- Codegenerierung aus Aufgabenbeschreibungen
- Code-Refactoring und Optimierung
- Test-case Generierung und Validierung

- Fehlererklärungen und Debugging
- Assistenten für die Dokumentation
- Gepaarte Programmier-Copiloten

Implementierungsleitfaden

- Sie können dieses Muster mit den folgenden Tools erstellen und: AWS-Services
- Amazon Bedrock für LLM-driven Generierung und Argumentation
- Amazon Q Developer für Vorschläge und Ergänzungen zur Codierung
- AWS Lambda oder Amazon Elastic Container Service (Amazon ECS) zum Ausführen und Testen von Sandbox-Umgebungen
- AWS Cloud9, VS Code-Erweiterungen oder benutzerdefinierte IDE-Integrationen zum Hosten und Auswerten von Kontext
- Amazon Simple Storage Service (Amazon S3) zum Speichern von Zwischenaufforderungen, Antworten und dem Versionsverlauf

Zusammenfassung

Coding Agents sind neue AI-powered Entwicklungstools, die in der Lage sind, natürliche Sprache zu interpretieren, den Kontext zu analysieren, mehrstufige Codeänderungen zu generieren und sich in den Softwareentwicklungszyklus zu integrieren.

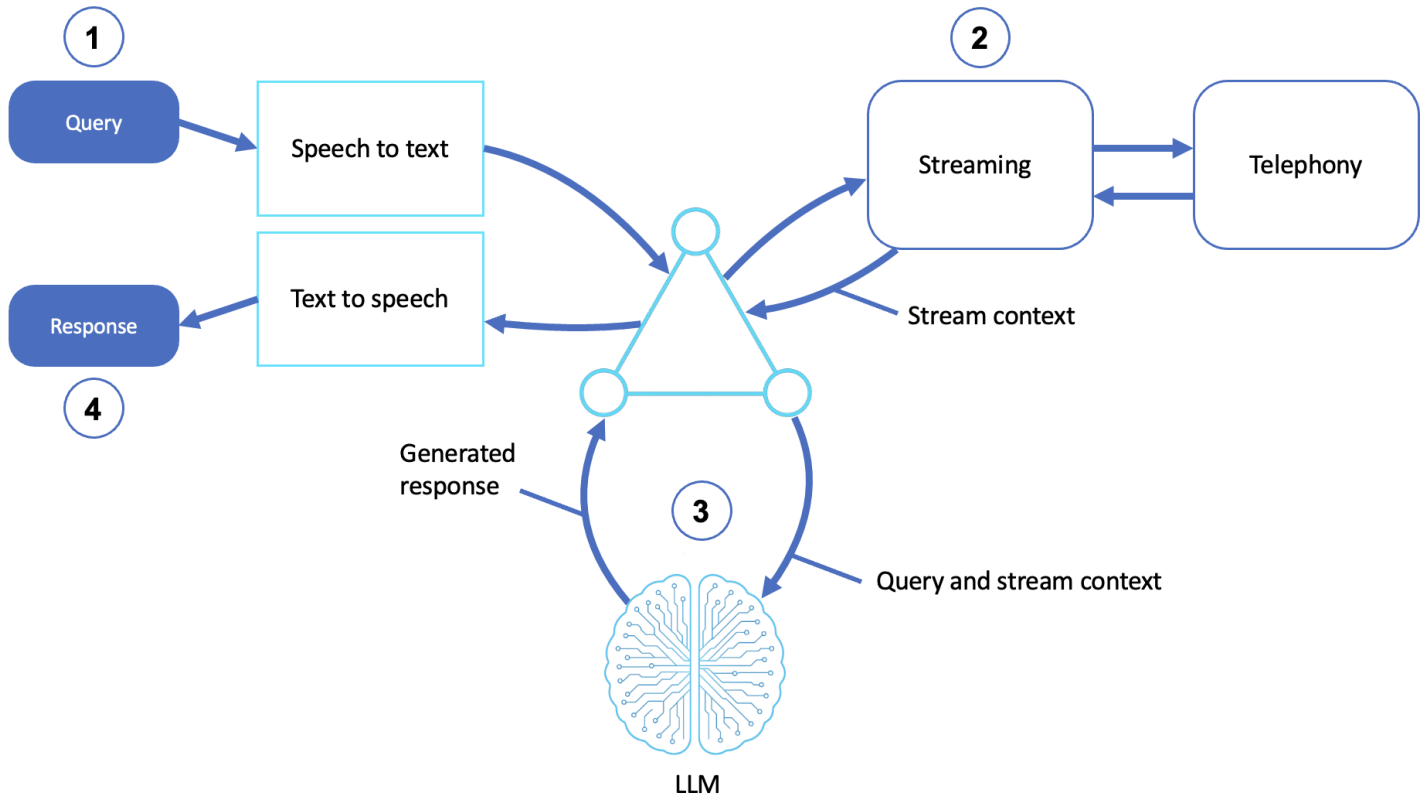
Sprach- und Sprachagenten

Sprach- und Sprachagenten interagieren mit Benutzern im Rahmen eines gesprochenen Dialogs. Diese Agenten integrieren Spracherkennung, natürliches Sprachverständnis und Sprachsynthese, um Konversations-KI auf Telefonie-, Mobil-, Web- und eingebetteten Plattformen zu ermöglichen.

Sprachagenten sind besonders effektiv in Umgebungen mit Freisprechfunktion, Echtzeit- oder Barrierefreiheit. Durch die Kombination von Streaming-Benutzeroberflächen mit LLM-powered Argumentation ermöglichen sie umfangreiche, dynamische Interaktionen, die sich für Benutzer natürlich anfühlen.

Architektur

In der folgenden Abbildung ist ein Sprach- und Sprachagent dargestellt:



Description

1. Empfängt eine Sprachanfrage
   - Der Benutzer stellt eine Anfrage an ein Telefon, ein Mikrofon oder ein eingebettetes System.
   - Ein Sprache-zu-Text-Modul (STT) konvertiert das Audio in Text.
2. Integriert Streaming- und Telefoniekontext
   - Der Agent verwendet eine Streaming-Schnittstelle, um Audio I/O in Echtzeit zu verwalten.
   - Wenn sie in einem Contact Center oder im Telekommunikationskontext eingesetzt wird, übernimmt die Telefonieintegration das Sitzungsrouting, die DTMF-Eingabe (Dual-Tone Multifrequency) und den Medientransport.

Hinweis: DTMF bezieht sich auf die Töne, die erzeugt werden, wenn Sie Tasten auf einer Telefontastatur drücken. Im Zusammenhang mit der Integration von Streaming- und Telefoniekontexten innerhalb von Sprachagenten wird DTMF als Signaleingabemechanismus während eines Telefonanrufs verwendet, insbesondere in Interactive Voice Response (IVR) - Systemen. DTMF-Eingaben ermöglichen dem Agenten:

- Erkennen Sie Menüauswahlen (z. B. „Drücken Sie 1 für die Abrechnung. Drücken Sie 2, um Unterstützung zu erhalten.“)
- Erfassen Sie numerische Eingaben (z. B. Kontonummern, PINs und Bestätigungsnummern)
- Lösen Sie Workflows oder Zustandsübergänge in Anrufabläufen aus
- Wechseln Sie bei Bedarf von der Sprach- zur Tonwiedergabe

1. Gründe dafür sind der LLM-Stream-Kontext
   - Die Abfrage wird an den Agenten gesendet, der sie zusammen mit allen Sitzungsmetadaten (z. B. Anrufer-ID, vorheriger Kontext) an ein LLM weiterleitet.
   - Das LLM generiert eine Antwort, möglicherweise mithilfe einer Chain-of-Thought-Strategie oder eines Multiturn-Memory, wenn die Interaktion andauert.
2. Gibt eine Sprachantwort zurück
   - Der Agent wandelt seine Antwort mithilfe von Text-to-Speech (TTS) in Sprache um.
   - Er gibt Audio über einen Sprachkanal an den Benutzer zurück.

Capabilities

- Real-time Sprachverständnis und Sprachgenerierung
- Mehrsprachig I/O mit STT- und TTS-Unterstützung
- Integration mit Telefonie- oder Streaming-APIs
- Sitzungsbewusstsein und Speicherübergabe zwischen den Runden

Häufige Anwendungsfälle

- IVR-Systeme für Konversationen
- Virtuelle Rezeptionisten und Terminplaner
- Voice-driven Helpdesk-Agenten
- Tragbare Sprachassistenten
- Sprachschnittstellen für Smart Homes und Tools zur Barrierefreiheit

Implementierungsleitfaden

Sie können dieses Muster mit den folgenden Tools erstellen und AWS-Services:

- Amazon Lex V2 oder Amazon Transcribe für STT
- Amazon Polly für TTS
- Amazon Chime SDK, Amazon Connect Customer oder Amazon Interactive Video Service (Amazon IVS) für Streaming und Telefonie
- Amazon Bedrock zum Argumentieren mit Anthropic, AI21 oder anderen Foundation-Modellen
- AWS Lambda um STT, LLM, TTS und den Sitzungskontext zu verbinden

(Optional) Zusätzliche Verbesserungen können Folgendes umfassen:

- Amazon Kendra oder OpenSearch für kontextsensitives RAG
- Amazon DynamoDB für Sitzungsspeicher
- Amazon CloudWatch Logs und AWS X-Ray zur Rückverfolgbarkeit

Zusammenfassung

Sprach- und Sprachagenten sind intelligente Systeme, die über natürliche Konversationen miteinander interagieren. Durch die Integration von Sprachschnittstellen mit LLM-Argumentation und Echtzeit-Streaming-Infrastruktur ermöglichen Sprachagenten nahtlose, zugängliche und skalierbare Interaktionen.

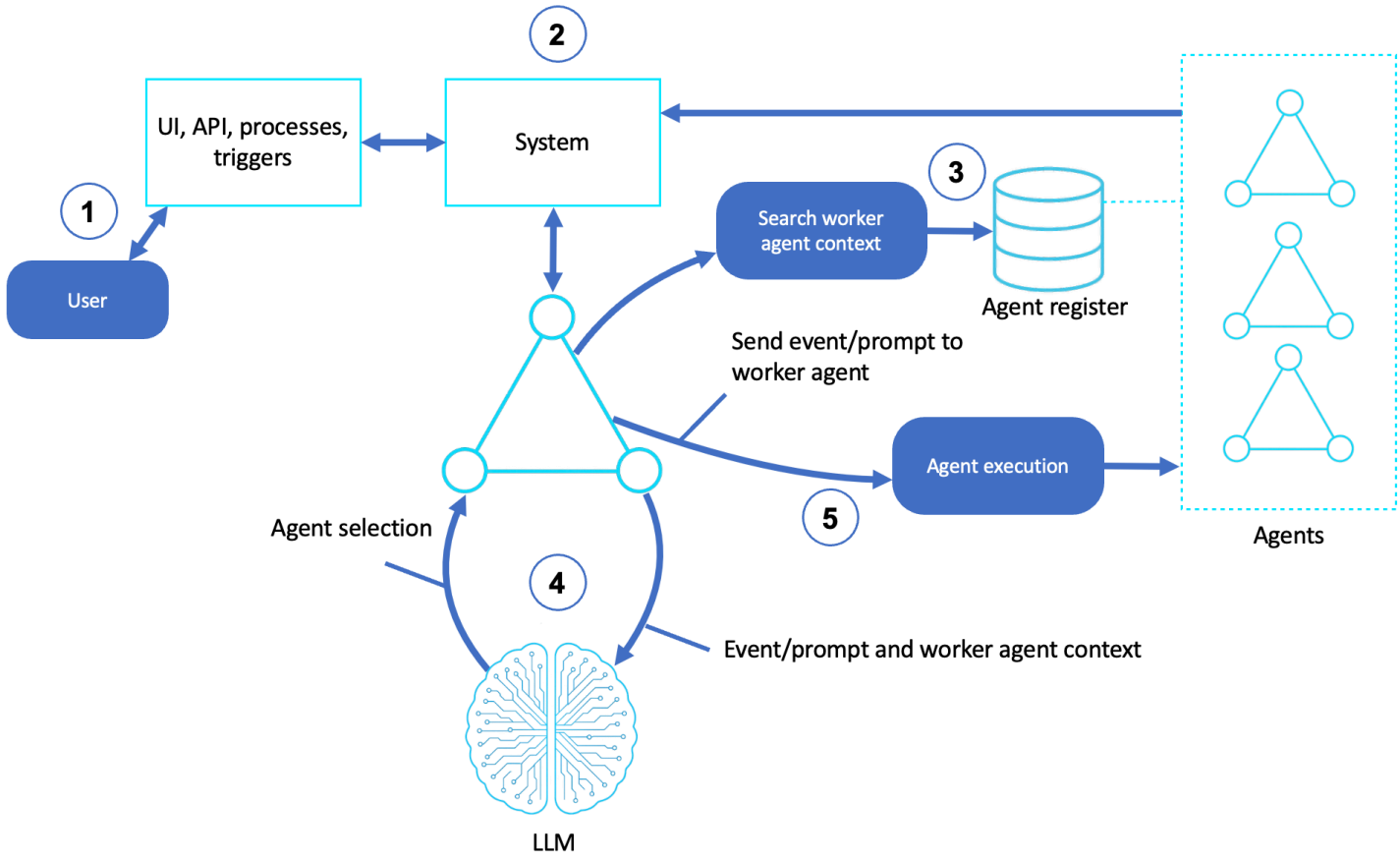
Agenten zur Workflow-Orchestrierung

Agenten zur Workflow-Orchestrierung verwalten und koordinieren mehrstufige Aufgaben, Prozesse und Dienste in verteilten Systemen. Anstatt isoliert zu argumentieren und zu handeln, delegieren diese Agenten Arbeit an Unteragenten oder andere Systeme, behalten den Ausführungskontext bei und passen sich auf der Grundlage von Zwischenergebnissen an.

Diese Agenten sind ein grundlegender Bestandteil von Automatisierungsabläufen. Sie sind besonders nützlich bei der Bearbeitung von Aufgaben mit langer Laufzeit, der Zusammenstellung mehrerer Agenten und domänenübergreifenden Integrationen, bei denen verschiedene Agenten und Tools nacheinander oder bedingt aufgerufen werden müssen.

Architektur

In der folgenden Abbildung ist ein Agent zur Workflow-Orchestrierung dargestellt:



Description

1. Empfängt Benutzereingaben
   - Ein Benutzer (oder ein externer Trigger) initiiert eine Aufgabe über eine Benutzeroberfläche, eine API oder ein Systemereignis.
2. Behandelt Systemereignisse
   - Eine Systemkomponente empfängt die Anforderung und gibt ein Ereignis oder einen Befehl aus, für das eine Orchestrierung erforderlich ist.
3. Ruft den Kontext ab
   - Der Workflow-Agent fragt Wissensdatenbanken und Agentenregister ab, um anhand von Metadaten, Domäne und früherer Erfolgsquote den richtigen Worker Agent für die Aufgabe zu finden.
4. Wählt einen LLM-Agenten aus

- Ein LLM hilft bei der Auswahl des besten Agenten oder Workflow-Plans, indem er die Aufgabenbeschreibung und die verfügbaren Optionen analysiert.
- Es kann auch aufgabenspezifische Aufforderungen formulieren, die an einen ausgewählten Agenten gesendet werden sollen.

5. Delegiert und führt aus
   - Der gewählte Worker-Agent empfängt das Ereignis oder die Aufforderung und beginnt mit der Ausführung von Befehlen.
   - Er kann den Ausführungsstatus verfolgen, bei einem Fehler den Vorgang wiederholen und Zwischenergebnisse an den nächsten Agenten in der Sequenz weiterleiten.

Capabilities

- Zusammensetzung der Agenten (z. B. Vorgesetzte, Mitarbeiter und Tools)
- Event-driven oder geplante Ausführung
- Speicher- und Statusverfolgung im Zeitverlauf
- Hierarchische oder parallel Aufgabenorchestrierung (synchron im Vergleich zu asynchronen Workflows)
- Dynamische Agentenauswahl und Verkettung

Häufige Anwendungsfälle

- Mehrstufige Automatisierung (z. B. Datenerfassung und Berichterstattung)
- Weiterleitung und Eskalation des Kundendienstes (z. B. „Agent-as-Coordinator“)
- KI-Agenten koordinieren sich mit Menschen und Bots innerhalb derselben Schleife
- Automatisiert Unternehmensprozesse mithilfe von Logik LLM-powered
- Hybridsysteme kombinieren KI-Agenten und traditionelle Orchestrierungstools

Implementierungsleitfaden

Sie können dieses Muster mit den folgenden Tools erstellen und AWS-Services:

- Amazon Bedrock zur Argumentation und Agentenauswahl
- AWS Step Functions oder Amazon EventBridge für die Workflow-Zusammenstellung

- AWS Lambda als Ausführungseinheiten oder Task-Runner
- Amazon DynamoDB, Amazon Simple Storage Service (Amazon S3) oder Amazon RDS zur Nachverfolgung von Status und Ergebnissen
- AWS AppFabric oder Amazon AppFlow für die systemübergreifende Koordination
- (Optional) Verwenden Sie Amazon SageMaker Run Agent, um domänenspezifische Worker-Agents zu hosten

Zusammenfassung

Workflow-Agenten koordinieren, passen sich an und stimmen Ziele in Umgebungen mit mehreren Agenten ab. Das bedeutet, dass KI-Agenten zusammenarbeiten, sich an die Laufzeitbedingungen anpassen und mithilfe modularer, erklärbarer Workflows komplexe Ergebnisse erzielen können.

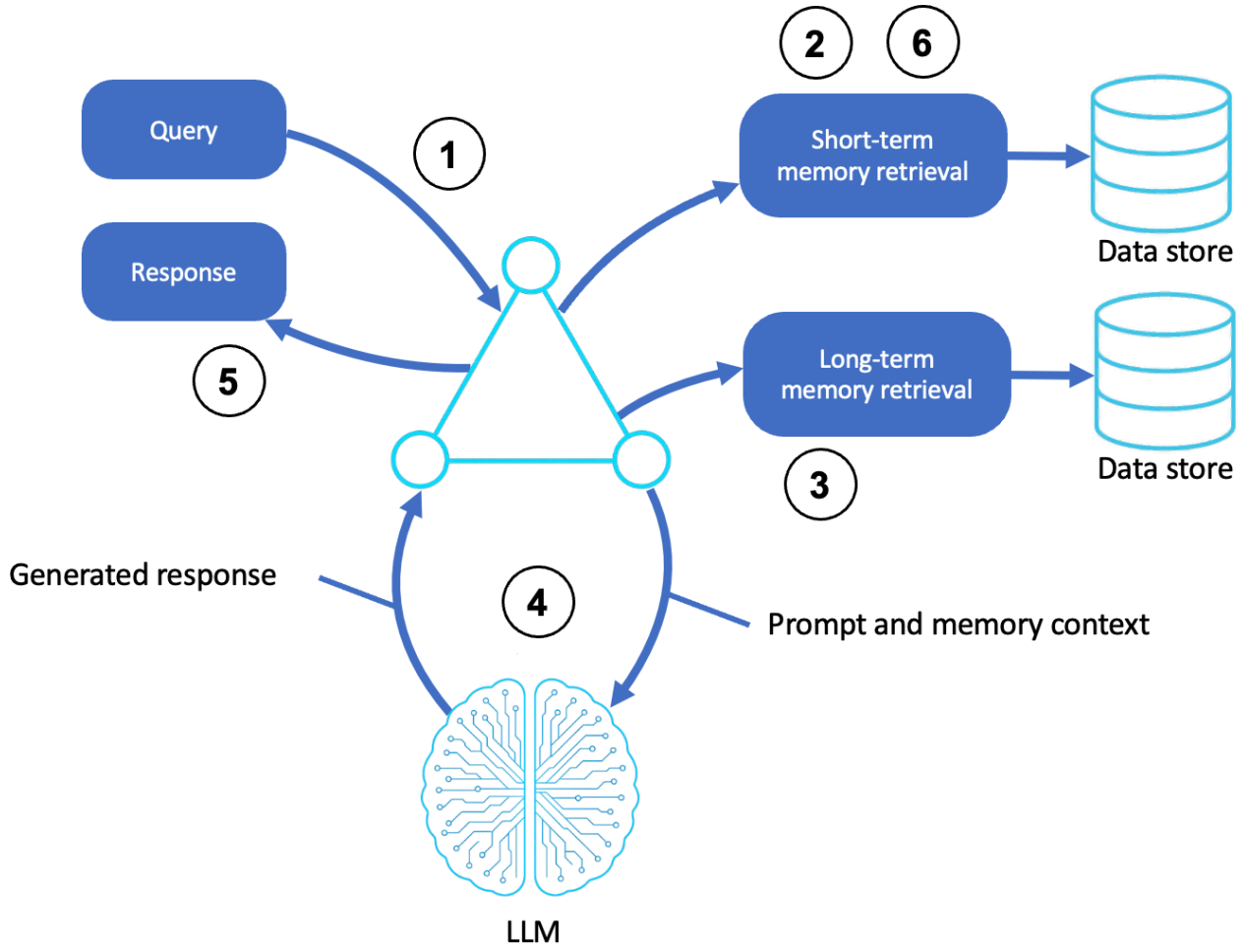
Memory-augmented Agenten

Memory-augmented Agenten sind nun in der Lage, das Kurz- und Langzeitgedächtnis zu speichern, abzurufen und zu analysieren. Auf diese Weise können sie den Kontext mehrerer Aufgaben, Sitzungen und Interaktionen aufrechterhalten, was zu kohärenteren, personalisierteren und strategischeren Antworten führt.

Im Gegensatz zu Agenten ohne Status passen sich Agenten mit erweitertem Speicher an, indem sie auf historische Daten zurückgreifen, aus früheren Ergebnissen lernen und Entscheidungen treffen, die den Zielen, Vorlieben und der Umgebung des Benutzers entsprechen.

Architektur

Ein Agent mit erweitertem Speicher ist in der folgenden Abbildung dargestellt:



Description

1. Empfängt Eingaben oder Ereignisse
   - Der Agent empfängt eine Benutzerabfrage oder ein Systemereignis. Dies kann ein Text, ein API-Trigger oder eine Änderung der Umgebung sein.
2. Ruft Kurzzeitgedächtnis ab
   - Der Agent ruft den aktuellen Gesprächsverlauf, den Aufgabenkontext oder den Systemstatus ab, der für die Sitzung oder den Workflow relevant ist.
3. Ruft das Langzeitgedächtnis ab
   - Der Agent fragt das Langzeitgedächtnis (z. B. Vektordatenbanken und Schlüsselwertspeicher) ab, um historische Erkenntnisse wie die folgenden zu erhalten:
     - Benutzereinstellungen
     - Frühere Entscheidungen und Ergebnisse

- Gelernte Konzepte, Zusammenfassungen oder Erfahrungen

4. Begründungen durch das LLM
   - Der Speicherkontext ist in die LLM-Eingabeaufforderung eingebettet, sodass der Agent sowohl auf der Grundlage aktueller Eingaben als auch auf Vorkenntnissen argumentieren kann.
5. Generiert Ausgaben
   - Der Agent erstellt eine kontextsensitive Antwort, einen Plan oder eine Aktion, die entsprechend dem Aufgabenverlauf und den Eingaben des Benutzers personalisiert ist.
6. Aktualisiert den Speicher
   - Neue Informationen, wie z. B. aktualisierte Ziele, Erfolgs- und Fehlschlagssignale und strukturierte Antworten, werden für future Aufgaben gespeichert.

Capabilities

- Kontinuität der Sitzung über Konversationen oder Ereignisse hinweg
- Beharrlichkeit des Ziels im Laufe der Zeit
- Kontextuelles Bewusstsein auf der Grundlage eines sich entwickelnden Zustands
- Anpassungsfähigkeit basiert auf früheren Erfolgen und Misserfolgen
- Personalisierung, abgestimmt auf Benutzerpräferenzen und -historie

Häufige Anwendungsfälle

- Konversations-Copiloten, die sich Benutzerpräferenzen merken
- Codierungsagenten, die Änderungen an der Codebasis verfolgen
- Workflow-Agenten, die sich je nach Aufgabenverlauf anpassen
- Digitale Zwillinge, die sich aus Systemwissen entwickeln
- Recherchieren Sie Agenten, die redundante Abfragen vermeiden

Implementierung von Agenten mit erweitertem Speicher

Verwenden Sie die folgenden Tools und AWS-Services für Agenten mit erweitertem Speicher:

Speicherschicht	AWS-Service	Zweck

Short-term	Amazon DynamoDB-, Redis- und Amazon Bedrock-Kontext	Schnelles Abrufen der letzten Interaktionsstatus
Long-term (strukturiert)	Amazon Aurora, Amazon DynamoDB, Amazon Neptune	Fakten, Beziehungen und Protokolle
Long-term (semantisch)	OpenSearch, PostgreSQL, Tannenzapfen	Embedding-based Abruf (das heißt, RAG)
Speicher	Amazon S3	Speichern von Transkripten, strukturierten Speichern und Dateien
Orchestrierung	AWS Lambda oder AWS Step Functions	Verwaltung der Speicheri njektion und des Aktualisi erungszyklus
Reasoning	Amazon Bedrock	Anthropic Claude oder Mistral mit Erinnerungsaufford erungen

Implementierung von Eingabeaufforderungen mit Speichereingabe

Um das Gedächtnis in die Argumentation der Agenten zu integrieren, verwenden Sie eine Kombination aus strukturiertem Zustand und abruf-erweiterter Kontext-Injektion:

- Beziehen Sie bei der Erstellung der Eingabeaufforderung für das Sprachmodell den aktuellen Status des Agenten und den Verlauf der letzten Dialoge als strukturierte Eingabe mit ein, sodass der Mitarbeiter mit vollständigem Kontext argumentieren kann.
- Verwenden Sie Retrieval-Augmented Generation (RAG), um relevante Dokumente oder Fakten aus dem Langzeitgedächtnis abzurufen.
- Fassen Sie frühere Pläne, Kontexte und Interaktionen zusammen, um sie zu komprimieren und relevant zu machen.
- Fügen Sie während der Inferenz externe Speichermodule wie Vektorspeicher oder strukturierte Protokolle ein, um die Entscheidungsfindung zu unterstützen.

Zusammenfassung

Memory-augmented Agenten sorgen für Kontinuität im Denken, indem sie aus Erfahrungen lernen und sich den Benutzerkontext merken. Diese Agenten übertreffen reaktive Intelligenz, indem sie auf langfristige Zusammenarbeit, Personalisierung und strategisches Denken setzen. In Bezug auf agentische KI ermöglicht das Gedächtnis den Agenten, sich eher wie adaptive digitale Gegenstücke und weniger wie zustandslose Tools zu verhalten.

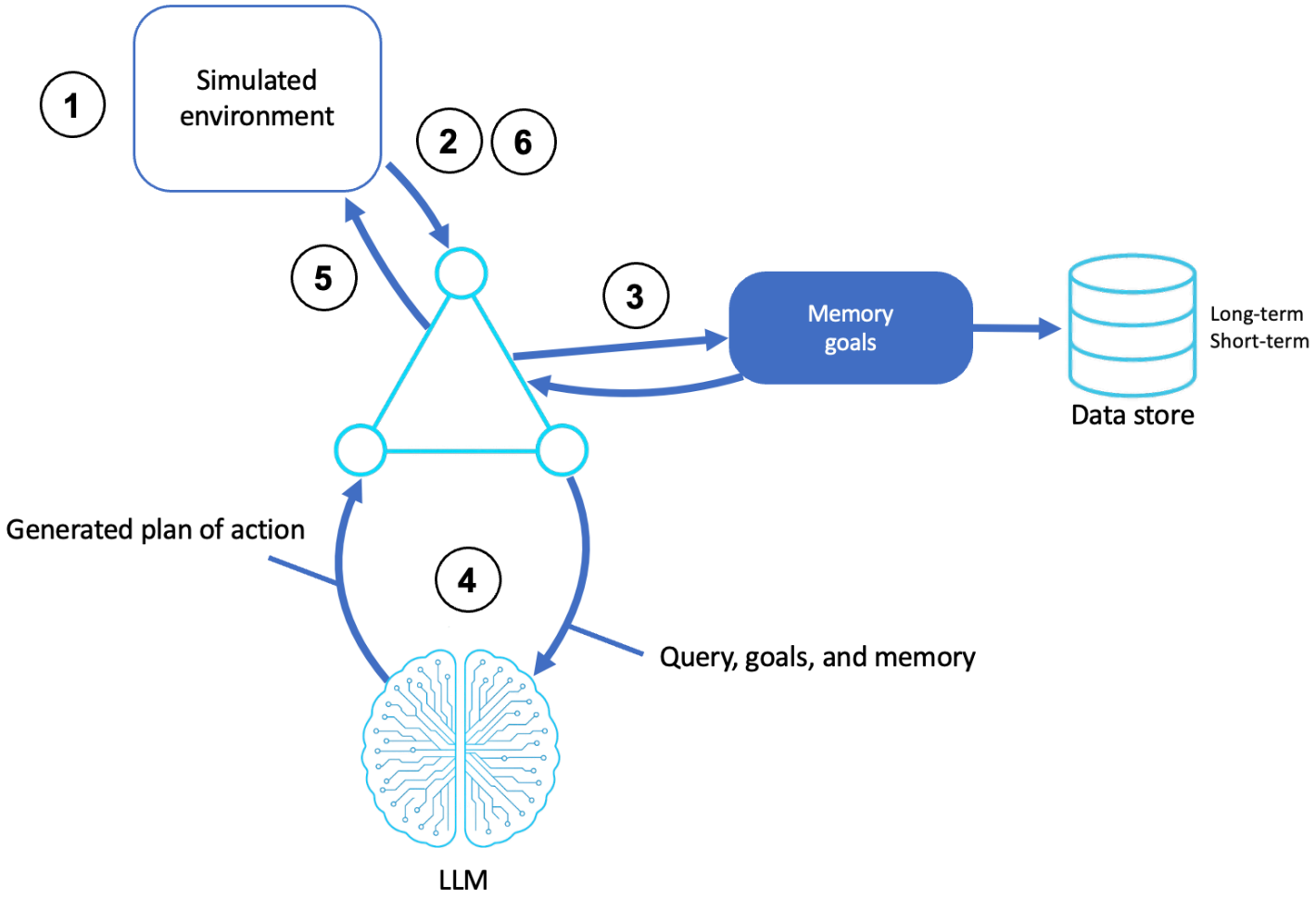
Agenten für Simulation und Testumgebung

Simulations- und Testbed-Agenten arbeiten in virtualisierten oder kontrollierten Umgebungen, in denen sie denken, handeln und lernen. Diese Agenten simulieren Verhalten, modellieren Ergebnisse und trainieren Strategien in wiederholbaren Umgebungen, bevor sie sie in realen Umgebungen anwenden.

Dieses Muster ist nützlich für iterative Entwicklung, verstärkendes Lernen (RL), die Bewertung autonomer Entscheidungen und das Testen von emergentem Verhalten. Simulationsagenten arbeiten oft in geschlossenen Kreisläufen, erhalten Feedback von ihrer Umgebung und passen ihr Verhalten entsprechend an. Deshalb sind sie unverzichtbar für Aufgaben, die räumliches Denken, Echtzeitsteuerung oder komplexe Systemdynamik beinhalten.

Architektur

Das folgende Diagramm zeigt einen Simulations- oder Testbed-Agenten:



Description

1. Initiiert eine Umgebung
   - Der Agent initiiert eine simulierte Umgebung (z. B. eine 3D-Welt, eine Physik-Engine, eine CLI-Sandbox oder einen synthetischen Datenstrom).
   - Der Agent wird mit einer ersten Aufgabe, einem Ziel oder einer Richtlinie in die Umgebung geladen.
2. Nimmt den Agenten wahr
   - Der Agent nimmt den aktuellen Status durch Simulationstelemetrie wahr (z. B. Sensoremulation, virtuelle Kamera und strukturierte Protokolle).
3. Ruft Ziel und Erinnerung ab
   - Der Agent ruft das ihm zugewiesene Ziel, seine Szenarioanweisungen oder sein kontextuelles Ziel ab.

- Er kann auch vorherigen Speicher abrufen, einschließlich der folgenden:
  - Long-term Strategien oder Richtlinien
  - Umweltkarten oder bekannte Einschränkungen
  - Frühere Erfolge oder Misserfolge ähnlicher Simulationen

4. Gründe und Pläne
   - Ein LLM interpretiert den simulierten Zustand, die Aufgabenziele und das erlernte Wissen.
   - Es generiert einen Aktionsplan oder einen Kontrollbefehl.
5. Führt simulierte Aktionen aus
   - Der Agent führt den Plan aus, ändert den Status, navigiert im Raum oder interagiert mit virtuellen Entitäten.
6. Lernt
   - Der Agent bewertet die Ergebnisse der Maßnahmen
   - Abhängig von der Konfiguration des Agenten kann er Folgendes tun:
     - Führen Sie RL aus
     - Protokollieren Sie die Ergebnisse für future Feinabstimmungen
     - Passen Sie Strategien in Echtzeit an

Capabilities

- Arbeitet in synthetischen oder virtuellen Umgebungen
- Unterstützt Lernen durch Versuch und Irrtum, die Verfeinerung von Richtlinien und die Systemmodellierung
- Low-risk Tests im Hinblick auf Verhalten, Fehlerbehandlung und Grenzfälle
- Ermöglicht die Analyse des Verhaltens neu auftretender Agenten in Konfigurationen mit mehreren Agenten
- Unterstützt sowohl die Steuerung im geschlossenen Regelkreis als auch die Erkundung durch den Menschen

Häufige Anwendungsfälle

- Reinforcement-Learning für Robotik, Drohnen und Spiele
- Training autonomer Fahrzeuge auf virtuellen Straßen

- Simulierte Benutzeroberflächen oder CLIs für Szenarien DevOps und Testsituationen
- Experimente mit neu auftretendem Verhalten in sozialen Simulationen
- Sicherheitsvalidierung der Entscheidungslogik vor der Produktion

Implementierungsleitfaden

Sie können einen Simulations- und Testbed-Agenten mit den folgenden Tools erstellen und: AWS-Services

Komponente	AWS-Service	Zweck
Umgebung	Amazon ECS, Amazon EC2 oder ein benutzerdefinierte r Simulator im Amazon SageMaker Studio Lab	Führen Sie virtuelle Welten (Gazebo, Unity, Unreal) oder Sandbox-CLIs aus
Agentenlogik	Amazon Bedrock SageMaker, Amazon oder AWS Lambda	LLM-based Planer oder RL-Agenten
Feedback-Schleife	Amazon SageMaker Reinforce ment-Learning CloudWatc h, Amazon oder benutzerd efinierte Protokolle	Nachverfolgung von Prämien, Ergebnisbewertung und Verhaltensprotokollierung
Erinnerung und Wiederholung	Amazon S3, Amazon DynamoDB oder Amazon RDS	Dauerhafte Status-, Episodenv erlaufs- oder Szenariodaten
Visualisierung	CloudWatch Amazon-Da shboards oder Amazon-No tizbücher SageMaker	Beobachten Sie politische Änderungen, Ergebnisse und Schulungskennzahlen

Im Folgenden sind zusätzliche Anwendungen aufgeführt:

- AWS SimSpace Weaverfür groß angelegte räumliche Simulationen
- AWS IoT Corezum Testen von Schattengeräten
- Amazon SageMaker Experiments zur Bewertung und zum Benchmarking von Agenten

Zusammenfassung

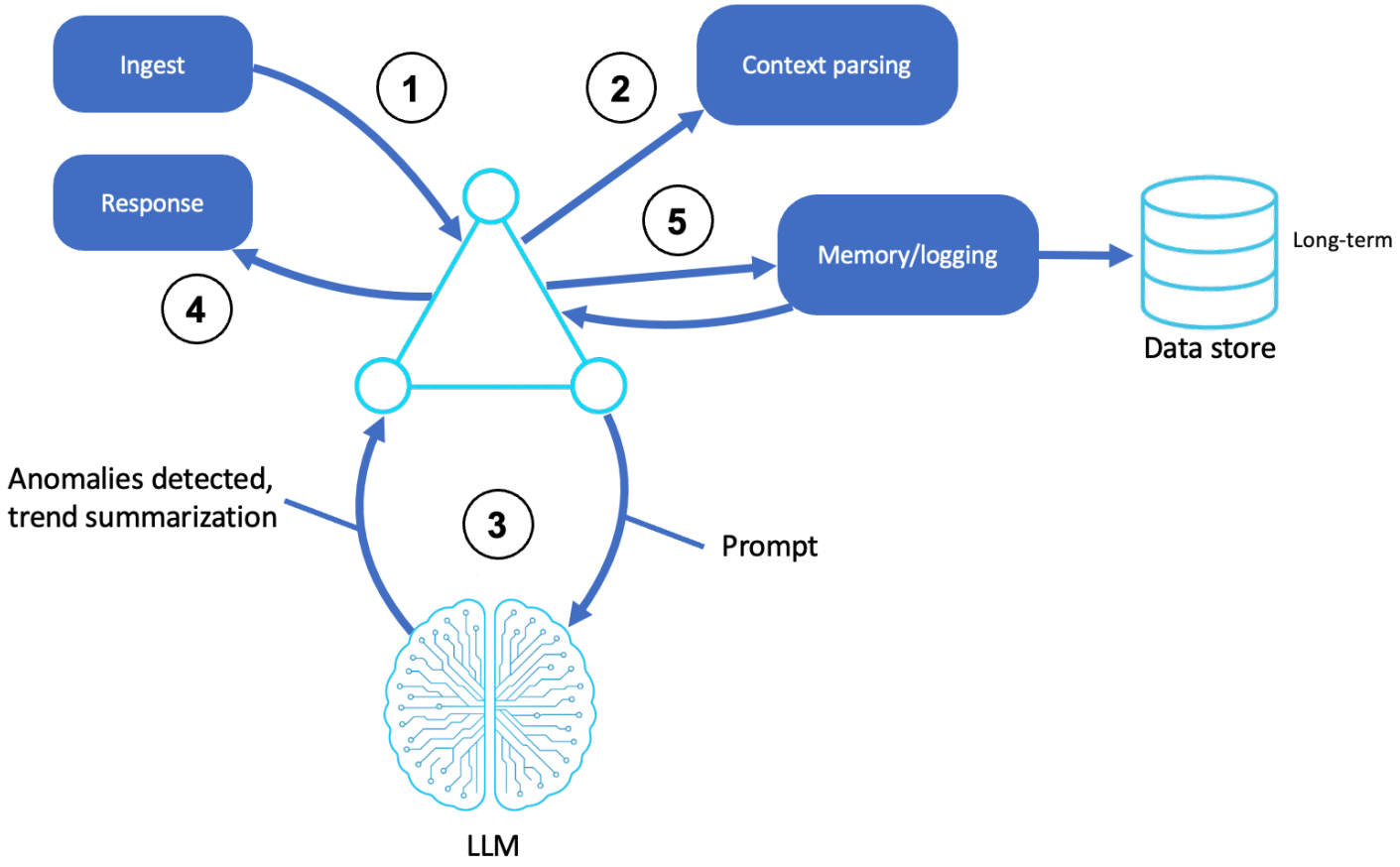
Simulations- und Prüfstandsagenten dienen der strukturierten Erkundung, bevor sie in Produktionssystemen eingesetzt werden. Verwenden Sie diese Agenten, um autonome Navigationsrichtlinien zu trainieren, Geschäftsprozesse in synthetischen Umgebungen zu testen und Schwärme auf Koordinationsmuster zu untersuchen.

Beobachter- und Überwachungsagenten

Beobachter- und Monitoring-Agenten beobachten passiv Systeme, Umgebungen und Interaktionen, um Muster zu erkennen, Erkenntnisse zu gewinnen und Aktionen auszulösen. Als intelligente Beobachter verbessern sie Warnmeldungen, Diagnosen und Audits, ohne direkt ein Verhalten auszulösen.

Diese Agenten zeichnen sich dort aus, wo es der herkömmlichen Überwachung an Anpassungsfähigkeit oder Argumentation mangelt, insbesondere in Bezug auf AI-in-the-loop Überwachung, Erkennung von Anomalien, Compliance-Überwachung und Sicherheitsinformationen. Observer Agents sind Ereignis-Listener, die Systemtelemetrie und Benutzerinteraktionen kontinuierlich überwachen. Der Agent ist auf seine Wahrnehmung, Interpretation und bedingte Eskalation oder Berichterstattung angewiesen.

Architektur



Description

1. Telemetrie aufnehmen
   - Der Agent empfängt Eingaben von einer oder mehreren Systemquellen, z. B. aus den folgenden:
     - Protokolle (Anwendung, Infrastruktur, Sicherheit)
     - Metriken (Leistung, Latenz, Nutzung)
     - Ereignisse (API-Aufrufe, Benutzeraktionen, Sensordaten)
2. Kontext analysieren
   - Rohdaten werden analysiert, strukturiert und mit Metadaten wie Zeitstempel, Akteuridentität, Systemstatus und Trace-ID angereichert.
3. Gründe für die Verwendung von LLM

- Der Agent verwendet ein LLM- oder Logikmodul, um analysierte Eingaben zu interpretieren, indem er Anomalien identifiziert, Trends zusammenfasst und Korrelationen zwischen verteilten Traces oder Zeitfenstern erstellt.

4. Klassifizieren oder warnen
   - Der Agent stellt fest, ob das beobachtete Verhalten Folgendes rechtfertigt:
     - Eine Warnung oder Eskalation
     - Ein Bericht oder ein Dashboard-Update
     - Ein Reaktionsauslöser (z. B. automatische Problembehebung und Richtliniendurchsetzung)
5. Speicher- oder Feedback-Schleifen protokollieren
   - Der Agent speichert Ereignisse und Entscheidungen für langfristiges Lernen, Audits oder future Referenzzwecke für andere Agenten.

Capabilities

- Passiv und nicht invasiv (der Agent wirkt nicht direkt)
- Hochgradig skalierbar und asynchron
- AI-driven Korrelation zwischen verrauschten oder verteilten Signalen
- Unterstützt Audits, Compliance und Einblicke in Echtzeit
- Kann nachgeschaltete Mitarbeiter oder menschliche Arbeitsabläufe unterstützen

Häufige Anwendungsfälle

- AI-augmented Beobachtbarkeit für Microservices und APIs
- Überwachung von Modellabweichungen, Richtlinienverstößen oder Out-of-Band-Verhalten
- Analyse der Kundenaktivitäten oder Zusammenfassungen der Interaktionen
- Code-Review-Agenten, die Commits oder Deployments überwachen
- Überwachung von Sicherheits- oder Compliance-Protokollen mithilfe von LLM-Argumentation

Implementierungsleitfaden

Sie können mit den folgenden Tools einen Beobachter- und Monitoring-Agenten erstellen: AWS-Services

Komponente	AWS-Service	Zweck
Erfassung von Ereignissen	Amazon EventBridge, CloudWatch Amazon-Protokolle, Amazon Kinesis, Amazon S3	Erfassen Sie strukturierte und unstrukturierte Telemetrie
Vorverarbeitung	AWS Lambda, AWS Glue, AWS Step Functions	Wandeln Sie Rohdaten in strukturierte Eingabeaufforderungen um
Maschine zum Argumentieren	Amazon Bedrock, Amazon SageMaker, AWS Lambda	Ereignisse analysieren, Verhalten klassifizieren, Erkenntnisse gewinnen
Speicher und Arbeitsspeicher	Amazon S3, Amazon DynamoDB, OpenSearch	Ständige Beobachtungen, Zusammenfassungen und Ergebnisse
Alarmierung und Eskalation	Amazon SNS AWS AppFabric, Amazon EventBridge	Lösen Sie nachgeschaltete Systeme oder Agenten aus

Im Folgenden sind zusätzliche Anwendungen aufgeführt:

- AWS Security Hub CSPMfür die Überwachung von Sicherheitsprotokollen
- Amazon Quick zur Visualisierung von Agentenausgaben

Zusammenfassung

Beobachter- und Monitoring-Agenten verfolgen Systeme und Verhalten in Echtzeit. Sie erkennen Anomalien, überprüfen die Sicherheit und sammeln Betriebsinformationen, indem sie Muster identifizieren, die von Menschen oder Regeln möglicherweise übersehen werden. Diese Fähigkeit hilft bei der Entwicklung von Systemen, die sich an sich ändernde Bedingungen anpassen und Entscheidungen auf der Grundlage umfassender Datenanalysen treffen können.

Multi-agent Zusammenarbeit

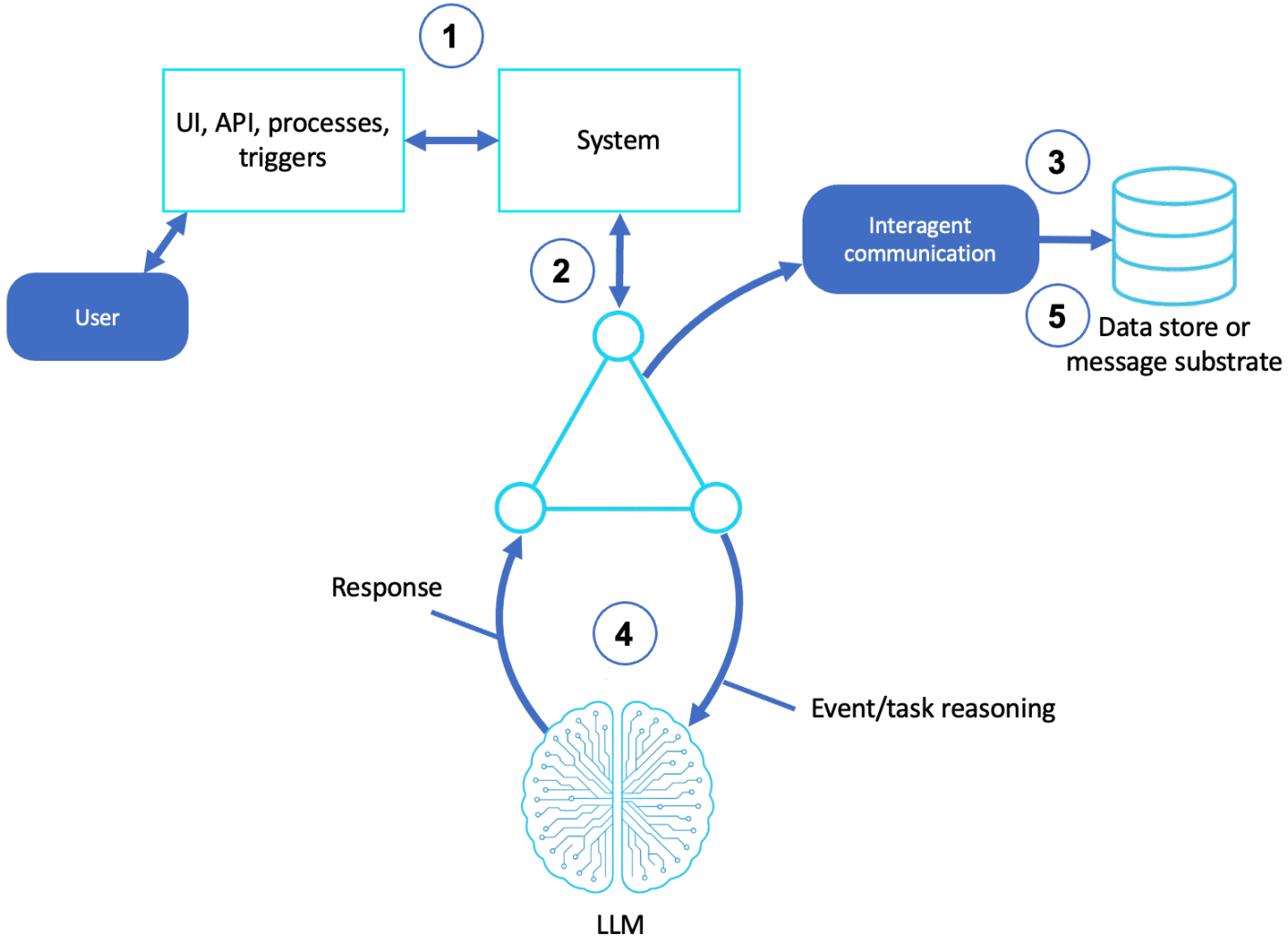
Multi-agent Zusammenarbeit bezieht sich auf ein Muster, bei dem mehrere autonome Akteure, von denen jeder eine eigene Rolle, Spezialisierung oder Zielsetzung hat, miteinander verhandeln, um komplexe Aufgaben zu lösen. Diese Agenten können unabhängig oder zusammen mit anderen Akteuren agieren, indem sie Informationen austauschen, Verantwortlichkeiten aufteilen und gemeinsam auf ein Ziel hinarbeiten.

Dieses Muster unterscheidet sich von Workflow-Agenten, die Aufgaben zentral koordinieren und in einem strukturierten Ablauf an untergeordnete Agenten delegieren. Im Gegensatz dazu betont die Zusammenarbeit mehrerer Agenten die Koordination zwischen Kollegen oder Emergente, indem Adaptivität, Parallelität und kognitive Teilung ermöglicht werden. In der folgenden Tabelle wird die Zusammenarbeit mehrerer Agenten mit Workflow-Agenten verglichen:

Funktion	Workflow-Agenten	Zweck
Steuerung	Zentralisierter Koordinator	Dezentrale, verteilte oder rollenbasierte Kollegen
Interaktionen	Ein Agent delegiert und verfolgt die Ausführung	Mehrere Agenten verhandeln, tauschen sich aus und passen sich an
Design	Vordefinierte Reihenfolge der Aufgaben	Emergente, flexible Aufgabenverteilung
Koordination	Prozedurale Orchestrierung	Kooperative oder kompetitive Interaktionen
Anwendungsfälle	Automatisierung von Unternehmensprozessen	Komplexes Denken, Erkundung und neue Strategien

Architektur

Das folgende Diagramm zeigt die Zusammenarbeit mehrerer Agenten:



Description

1. Leitet eine Aufgabe ein
   - Ein Benutzer oder ein System gibt ein übergeordnetes Ziel oder Problem aus.
   - Ein „Manager“, ein Agent oder ein initiierender Kontext definiert das Ziel.
2. Weist Rollen zu oder entdeckt sie
   - Agenten weisen sich selbst zu (symbolische Logik oder Argumentation) oder werden delegiert (Eventbroker) an andere Rollen, z. B. als Planer, Forscher, Testamentsvollstrecker, Kritiker oder Erklärer.
3. Kommuniziert mit anderen Agenten
   - Agenten kommunizieren über gemeinsamen Speicher, Nachrichtenwarteschlangen oder die Verkettung von Eingabeaufforderungen.

- Sie können sich gegenseitig diskutieren, Fragen stellen oder Unteraufgaben vorschlagen.

4. Verwendet spezielle Argumentation
   - Jeder Agent verwendet sein eigenes Modell oder seine eigene Domänenlogik, um seinen Teil des Problems zu lösen.
   - Agenten können LLMs mit rollenspezifischen Eingabeaufforderungen und Arbeitsspeicher verwenden.
5. Koordiniert Ergebnisse oder Ziele
   - Die Agenten fassen die Beiträge zu einer endgültigen Antwort, einem Plan oder einer Maßnahme zusammen.
   - (Optional) Ein Supervisor kann das synthetisierte Ergebnis validieren oder zusammenfassen.

Capabilities

- Peer-level Agenten mit speziellen Rollen oder Fähigkeiten
- Aufkommendes Verhalten durch Kommunikation oder Verhandlung
- Parallele Bearbeitung komplexer oder vielschichtiger Probleme
- Unterstützt Überlegung, Selbstkorrektur und reflektierende Iteration
- Modellieren Sie soziale Dynamiken, wissenschaftliche Zusammenarbeit oder Rollen in Unternehmensteams

Häufige Anwendungsfälle

- Autonome Forschungsteams (Suchagent, Summarizer und Validator)
- Softwareentwicklung (Planer, Programmierer und Tester)
- Modellierung von Geschäftsszenarien (Finanzen, Richtlinien und Compliance)
- Verhandlung, Angebotsabgabe oder Argumentation durch mehrere Parteien
- Multimodale Aufgaben (Bild, Text und Logik)

Implementierungsleitfaden

Sie können ein System mit mehreren Agenten mithilfe der folgenden Tools erstellen und: AWS-Services

Komponente	AWS-Service	Zweck
Agenten-Hosting	Amazon Bedrock, Amazon SageMaker, AWS Lambda	Hosten Sie einzelne Agenten LLM-driven
Kommunikationsebene	Amazon SQS, Amazon EventBridge, AWS AppFabric	Nachrichtenübermittlung und Koordination zwischen den Agenten
Gemeinsamer Speicher	Amazon DynamoDB, Amazon S3 oder OpenSearch	Multi-agent Speicher- oder Blackboard-System
Orchestrierungsebene	AWS Step Functions, AWS Lambda Rohrleitungen	Kickoff-, Timeout-, Fallback- und Wiederholungslogik
Identifizierung des Agenten	Amazon Bedrock Agents (rollendefiniert) AWS AppConfig und Amazon Bedrock Converse API (Agenten außerhalb von Amazon Bedrock)	Role-based Aufruf von Tools oder Agenten und Durchsetz ung von Grenzen
Emergente Interaktion	EventBridge Amazon-Pi pelines oder Agentenregister	Aktivieren Sie die dynamische Weiterleitung oder Eskalation von Aufgaben

Zusammenfassung

Multi-agent Durch die Zusammenarbeit werden die Aufgaben zur Problemlösung auf modulare, rollengesteuerte Agenten verteilt. Im Gegensatz zur Workflow-Orchestrierung nutzen Kooperationsmuster neue Intelligenz, Belastbarkeit und Skalierbarkeit, die die Art und Weise widerspiegeln, wie Menschen Probleme lösen. Dies ist besonders wertvoll für offene Bereiche, kreative Aufgaben, multimodales Denken und Umgebungen, die von unterschiedlichen Perspektiven profitieren.

Schlussfolgerung

Die zuvor erörterten Muster veranschaulichen grundlegende Ansätze für reale Implementierungen agentischer KI. Von grundlegenden Argumenten bis hin zu gedächtnisgestützter Intelligenz ist jedes Muster einzigartig für Wahrnehmung, Kognition und Handlung konfiguriert und basiert auf Autonomie, Asynchronität und Entscheidungsfreiheit.

Diese Muster haben gemeinsame Vokabeln und technische Pläne für den Aufbau intelligenter, zielgerichteter Systeme. Unabhängig davon, ob ein Muster in eine Benutzeroberfläche eingebettet, über Cloud-Dienste orchestriert oder von Agententeams koordiniert wird, jedes Muster ist anpassbar und modular.

Imbissbuden

- Agentenmuster sind zusammensetzbar — Die meisten Agenten aus der realen Welt kombinieren zwei oder mehr Muster (z. B. ein Sprachagent mit toolgestütztem Denken und Gedächtnis).
- Das Agentendesign ist kontextuell — Wählen Sie Muster auf der Grundlage der Interaktionsoberfläche, der Aufgabenkomplexität, der Latenztoleranz und domänenspezifischen Einschränkungen aus.
- AWSEine native Implementierung ist möglich — Mit Amazon Bedrock, Amazon SageMaker AWS Lambda AWS Step Functions, und ereignisgesteuerten Architekturen kann jedes Agentenmuster in großem Umfang bereitgestellt werden.

LLM-Workflows

Im Rahmen von Agentenmustern haben wir die gängigen Muster von KI-Agenten untersucht, die jeweils auf einer Reihe modularer Funktionen basieren: Wahrnehmung, Handlung, Lernen und Kognition. Das Herzstück des kognitiven Moduls vieler Agentenmuster ist ein großes Sprachmodell (LLM), das in der Lage ist, zu argumentieren, zu planen und Entscheidungen zu treffen. Der alleinige Aufruf eines LLM reicht jedoch nicht aus, um intelligentes, zielgerichtetes Verhalten zu erzeugen.

Um komplexe Aufgaben zuverlässig ausführen zu können, müssen Agenten das LLM in einen strukturierten Workflow einbetten, in dem die Funktionen des Modells durch Tools, Speicher, Planungsschleifen und Koordinationslogik erweitert werden. Diese LLM-Workflows ermöglichen es einem Agenten, Ziele aufzuschlüsseln, Unteraufgaben weiterzuleiten, externe Dienste in Anspruch zu nehmen, Ergebnisse zu analysieren und sich mit anderen Agenten abzustimmen.

In diesem Kapitel werden die wichtigsten Entwurfsmuster für die Erstellung robuster, erweiterbarer und intelligenter LLM-driven kognitiver Module vorgestellt, die auf wiederverwendbaren Workflows basieren.

In diesem Abschnitt

- Überblick über Kognition LLM-augmented
- Arbeitsablauf für schnelle Verkettung
- Arbeitsablauf für das Routing
- Workflow für die Parallelisierung
- Workflow für die Orchestrierung
- Arbeitsablauf für Evaluatoren und Reflect-Refine-Schleifen
- Schlussfolgerung

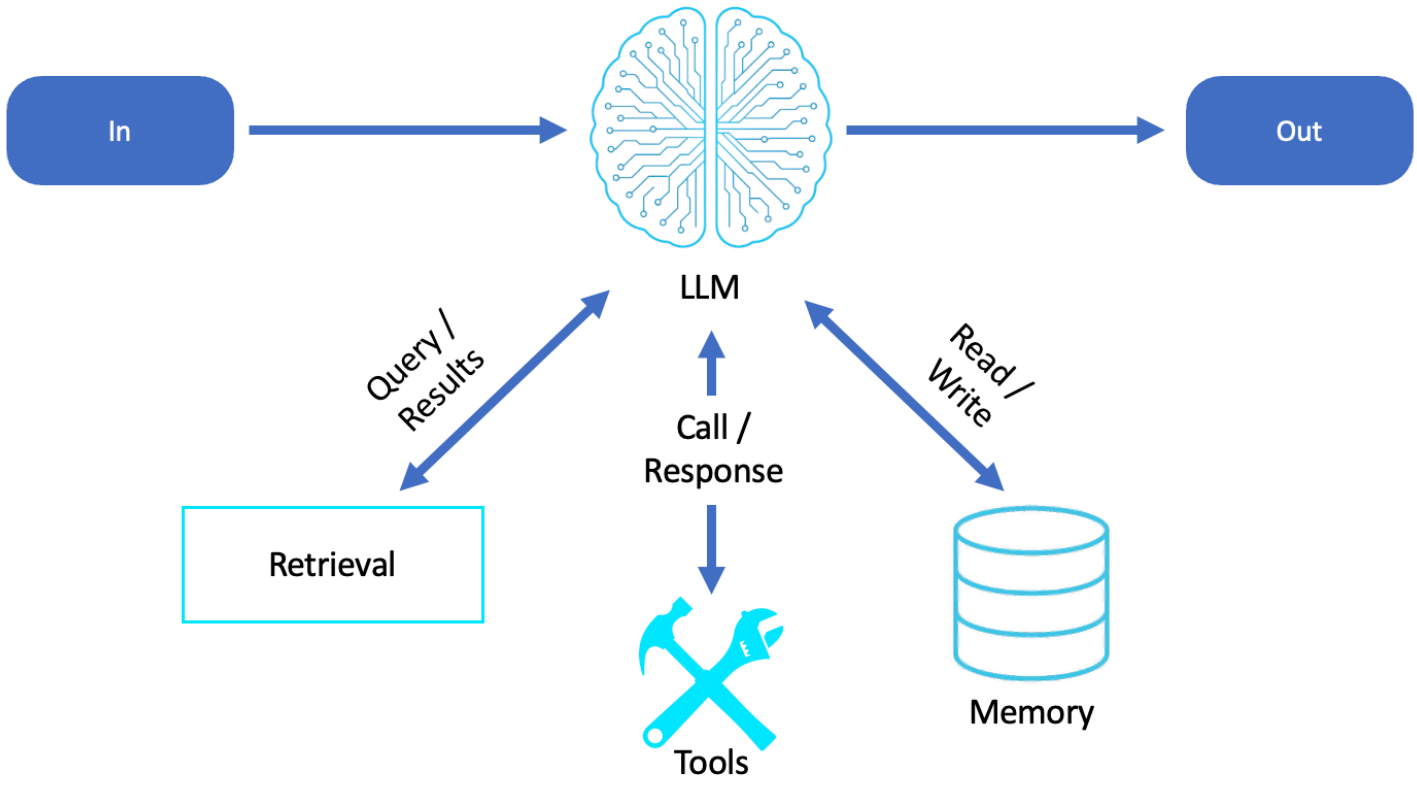
Überblick über Kognition LLM-augmented

Im Kern kann das kognitive Modul eines Softwareagenten als ein in Erweiterungen verpacktes LLM betrachtet werden. Der Agent kann die folgenden Bausteine verwenden, um in seiner Umgebung effektiv zu argumentieren:

- Eingabeaufforderung — Eingaben anhand von Kontext, Anweisungen, Beispielen und Speicher strukturieren

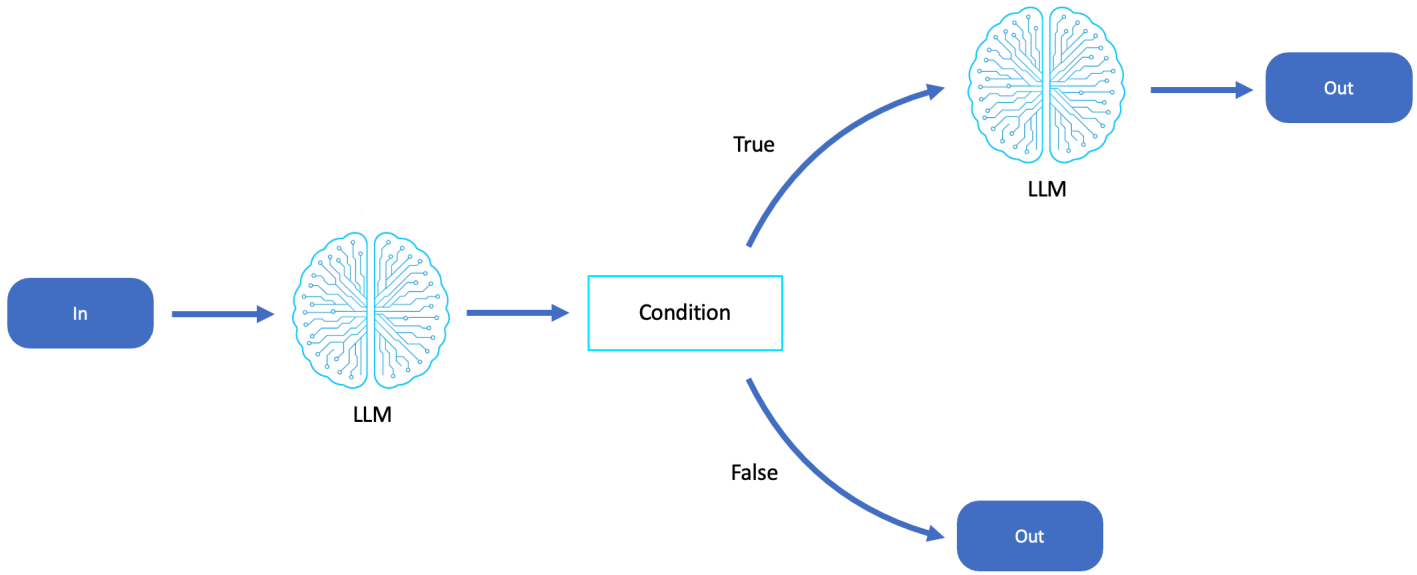
- Abruf — Bereitstellung von aktuellem oder domänenspezifischem Wissen für die LLM-Eingabeaufforderung durch Vektorsuche oder semantisches Gedächtnis, z. B. durch Retrieval-Augmented Generation (RAG)
- Verwendung von Tools — Ermöglicht es dem LLM, APIs aufzurufen oder Funktionen aufzurufen, um Informationen abzurufen oder darauf zu reagieren
- Speicher — Einbindung eines dauerhaften Zustands oder eines sitzungsbasierten Zustands in die Argumentationsschleife, entweder mithilfe strukturierter Datenbanken oder kontextueller Zusammenfassungen

Diese Erweiterungen bestehen aus Workflows, die definieren, wie das LLM im Laufe der Zeit und aufgabenübergreifend verwendet wird, wodurch es von einer zustandslosen Engine in einen dynamischen Argumentationsagenten umgewandelt wird.



Arbeitsablauf für schnelle Verkettung

Prompt Chaining zerlegt komplexe Aufgaben in eine Abfolge von Schritten, wobei jeder Schritt ein diskreter LLM-Aufruf ist, der die Ausgabe des vorherigen verarbeitet oder darauf aufbaut.



Der Workflow zur Verkettung von Eingabeaufforderungen eignet sich für Szenarien, in denen Aufgaben logisch in sequentielle Argumentationsschritte unterteilt werden können und in denen Zwischenergebnisse als Grundlage für die nächste Phase dienen. Er eignet sich hervorragend für Workflows, die strukturiertes Denken, fortschreitende Transformation oder mehrschichtige Analysen erfordern, wie z. B. die Überprüfung von Dokumenten, die Codegenerierung, die Extraktion von Wissen und die Verfeinerung von Inhalten.

Description

- Die Komplexität der Aufgabe übersteigt das Kontextfenster oder die Argumentationstiefe eines einzelnen LLM-Anrufs.
- Die Ergebnisse eines Schritts (z. B. Analyse, Zusammenfassung oder Planung) werden zu Inputs für eine Folgeentscheidungs- oder Generierungsphase.
- Sie benötigen Transparenz und Kontrolle in allen Phasen der Argumentation (z. B. überprüfbare Zwischenergebnisse).
- Sie möchten zwischen den Schritten eine externe Validierungs-, Filter- oder Anreicherungslogik einbauen.
- Es ist ideal für Agenten, die in Denkschleifen im Pipeline-Stil arbeiten, wie z. B. Rechercheagenten, Redaktionsassistenten, Planungssysteme und mehrstufige Copiloten.

Capabilities

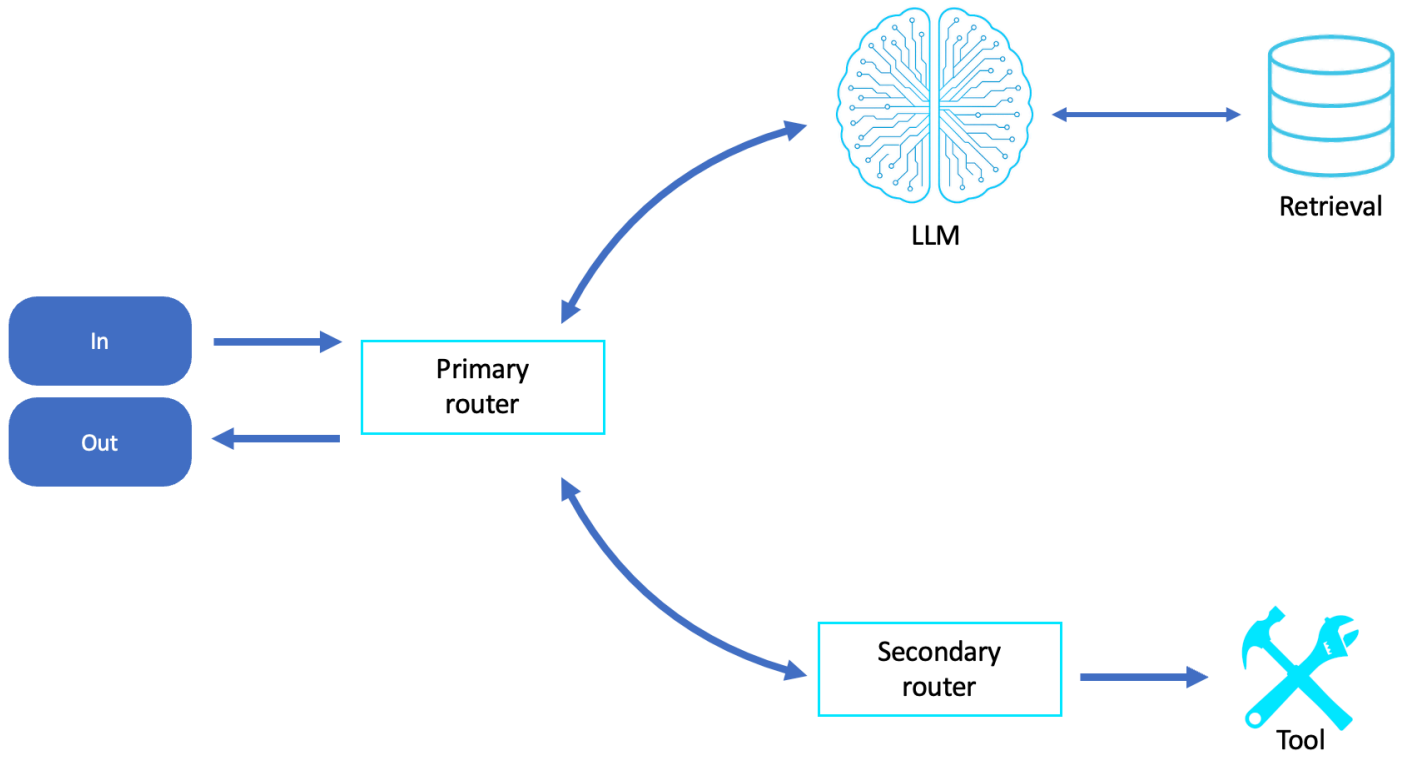
- Lineare oder verzweigte Ketten von LLM-Aufrufen
- Zwischenergebnisse wurden als strukturierte Eingabe weitergegeben oder in Folgeaufforderungen eingebettet
- Kann mit, oder agentenspezifischen AWS Step Functions Runnern AWS Lambda orchestriert werden

Häufige Anwendungsfälle

- Mehrstufige Aufgaben zum Argumentieren (z. B. „Kritik zusammenfassen, neu schreiben“)
- Wissenschaftliche Mitarbeiter, die mehrschichtige Ergebnisse zusammenfassen (z. B. „Suchen, Fakten extrahieren, Frage beantworten“)
- Pipelines zur Codegenerierung („Plan generieren, Code schreiben, Testcode erklären, Ausgabe erklären“)

Arbeitsablauf für das Routing

Im Routing-Muster verwendet ein Classifier oder Router-Agent ein LLM, um die Absicht oder Kategorie einer Abfrage zu interpretieren, und leitet die Eingabe dann an eine spezialisierte Downstream-Aufgabe oder einen Agenten weiter.



Der Routing-Workflow wird in Szenarien verwendet, in denen ein Agent die Eingabeabsicht, den Aufgabentyp oder die Domäne schnell klassifizieren und die Anfrage dann an einen speziellen Subagenten, ein Tool oder einen Workflow delegieren muss. Er ist besonders nützlich für Capability Agents, z. B. solche, die als allgemeine Assistenten, Eingangstüren zu Unternehmensfunktionen oder domänenübergreifende KI-Benutzeroberflächen für Benutzer dienen.

Routing ist besonders effektiv, wenn:

- Sortierung von Anfragen anhand einer Vielzahl von Aufgaben (z. B. Suche, Zusammenfassung, Buchung, Berechnungen).
- Eingaben müssen vorverarbeitet oder normalisiert werden, bevor sie in speziellere Workflows aufgenommen werden können.
- Verschiedene Eingabetypen (z. B. Bilder im Vergleich zu Text, strukturierte und unstrukturierte Abfragen) erfordern eine individuelle Handhabung.
- Ein Agent fungiert als Konversationszentrale und delegiert Aufgaben an spezialisierte Agenten oder Microservices.
- Dieser Workflow ist bei domänenspezifischen Copiloten, Bots für den Kundensupport, Enterprise Service Routern und multimodalen Agenten üblich, bei denen intelligentes Dispatching sowohl die Qualität als auch die Effizienz des Agentenverhaltens bestimmt.

Capabilities

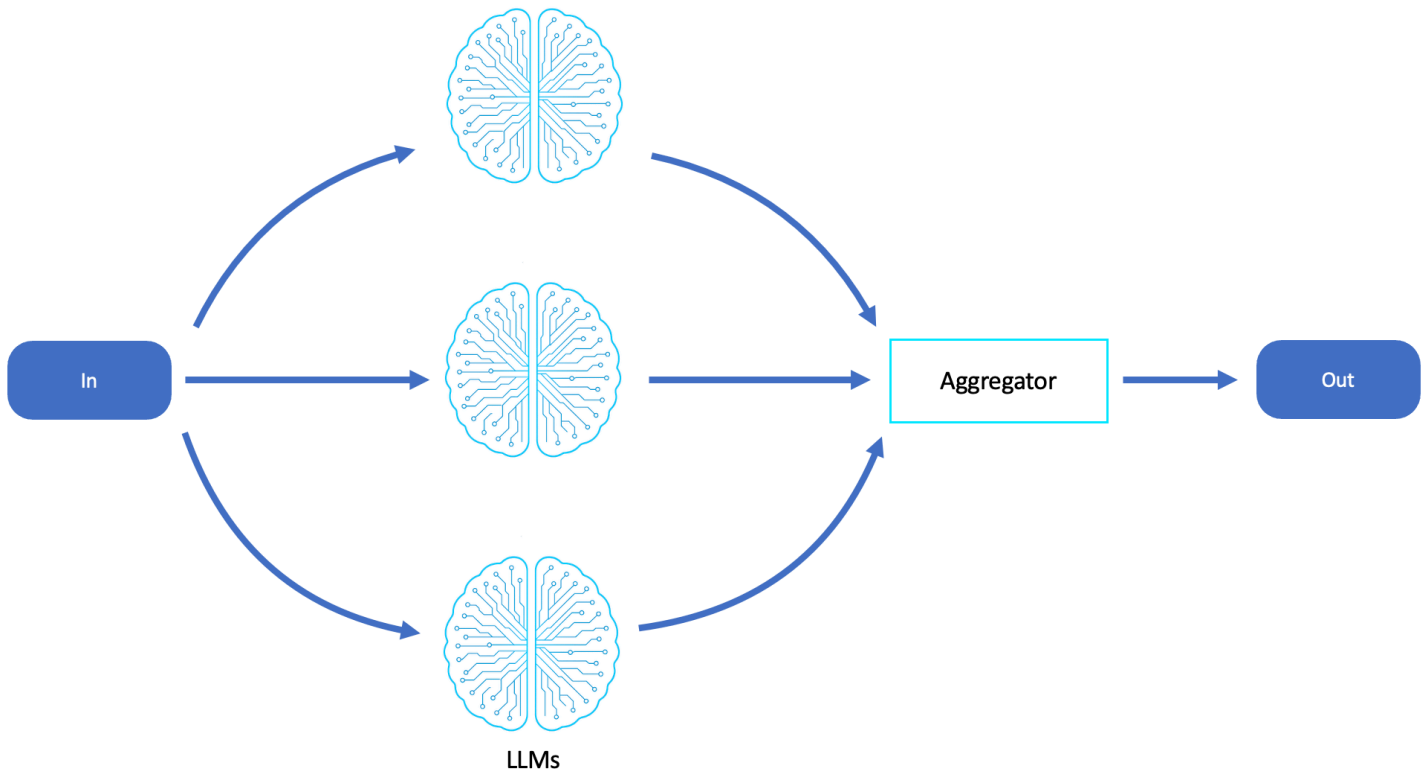
- Ein First-Pass-LM fungiert als Dispatcher
- Routen können unterschiedliche Workflows oder sogar andere Agentenmuster aufrufen
- Unterstützt die modulare Erweiterung von Funktionen

Häufige Anwendungsfälle

- Multidomain-Assistenten („Handelt es sich um eine rechtliche, medizinische oder finanzielle Frage?“)
- Die Entscheidungsbäume wurden durch LLM-Argumentation erweitert
- Dynamische Werkzeugauswahl (z. B. Suche im Vergleich zur Codegenerierung)

Workflow für die Parallelisierung

Dieser Workflow beinhaltet die Aufteilung einer Aufgabe in unabhängige Unteraufgaben, die gleichzeitig von mehreren LLM-Aufrufen oder Agenten bearbeitet werden können. Die Ergebnisse werden dann programmgesteuert aggregiert und zu einem Ergebnis zusammengefasst.



Der Parallelisierungs-Workflow wird verwendet, wenn eine Aufgabe in unabhängige, nicht sequentielle Unteraufgaben aufgeteilt werden kann, die gleichzeitig verarbeitet werden können, wodurch Effizienz, Durchsatz und Skalierbarkeit erheblich verbessert werden. Er ist besonders leistungsstark in datenintensiven, stapelorientierten oder multiperspektivischen Problembereichen, in denen der Agent Inhalte für mehrere Eingaben analysieren oder generieren muss.

Parallelisierung ist besonders effektiv, wenn:

- Teilaufgaben hängen nicht von den Zwischenergebnissen der jeweils anderen ab, sodass sie ohne Koordination parallel ausgeführt werden können.
- Eine Aufgabe besteht darin, denselben Argumentationsprozess für viele Aufgaben zu wiederholen (z. B. das Zusammenfassen mehrerer Dokumente oder das Auswerten einer Liste von Optionen).
- Mehrere Hypothesen oder Perspektiven werden parallel untersucht, um Vielfalt, Kreativität oder Robustheit zu fördern.
- Sie müssen die Latenz für Anfragen mit hohem Volumen oder hoher Frequenz durch gleichzeitige LLM-Ausführung reduzieren.
- Dieser Workflow wird häufig in Dokumentenverarbeitungsagenten, Umfrage- oder Vergleichs-Engines, Batchzusammenfassungen, Brainstormern mit mehreren Agenten und skalierbaren Klassifizierungs- oder Kennzeichnungsaufgaben verwendet, insbesondere dort, wo schnelles, paralleles Denken einen Leistungsvorteil darstellt.

Capabilities

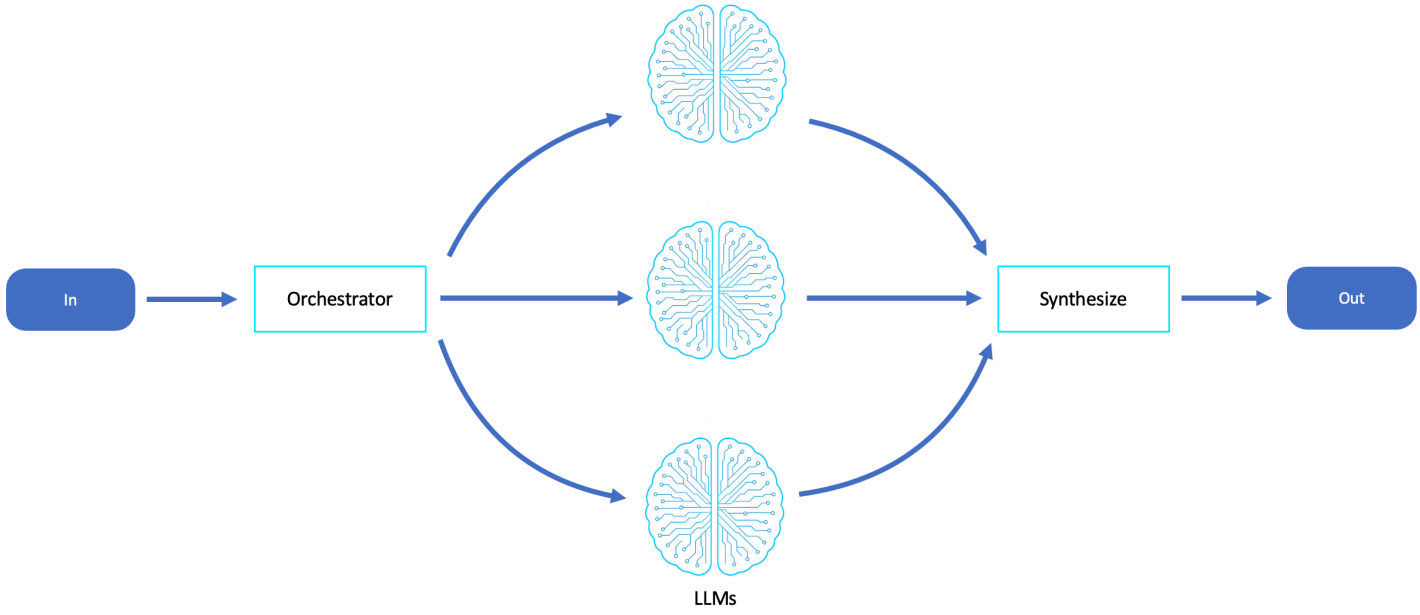
- Parallele Ausführung von LLM-Aufgaben (mithilfe von, oder einem Map-Status) AWS Lambda AWS Fargate AWS Step Functions
- Erfordert die Abstimmung, Validierung oder Deduplizierung der Ergebnisse in der Synthesephase
- Well-suited für statusfreie Agentenschleifen

Häufige Anwendungsfälle

- parallel Analyse mehrerer Dokumente oder Perspektiven
- Generierung verschiedener Entwürfe, Zusammenfassungen oder Pläne
- Beschleunigung des Durchsatzes bei Batch-Aufträgen

Workflow für die Orchestrierung

Ein zentraler Orchestrator-Agent verwendet ein LLM, um Unteraufgaben zu planen, zu zerlegen und an spezialisierte Worker Agents oder Modelle zu delegieren, die jeweils über eine bestimmte Rolle oder Fachkompetenz verfügen. Dies spiegelt die menschlichen Teamstrukturen wider und unterstützt aufkommendes Verhalten mehrerer Agenten.



Der Orchestrierungs-Workflow ist ideal für komplexe, hierarchische oder multidisziplinäre Szenarien, die eine strukturierte Zerlegung und spezielle Ausführung erfordern. Er eignet sich besonders gut für Aufgaben, die Arbeitsteilung erfordern und bei denen verschiedene Teilkomponenten einer Aufgabe am besten von Mitarbeitern mit unterschiedlichen Fähigkeiten, Kenntnissen oder Toolsets erledigt werden können.

Dieser Workflow ist besonders effektiv, wenn:

- Aufgaben können in Unteraufgaben unterteilt werden, die sich in Umfang, Art oder Argumentation unterscheiden (z. B. Planung, Recherche, Implementierung und Test).
- Ein LLM oder Metaagent muss andere Agenten koordinieren, den Fortschritt überwachen und die Ergebnisse zusammenfassen.
- Sie möchten die Zuständigkeiten der Agenten modularisieren, um Skalierbarkeit, Wiederverwendung und spezielle Feinabstimmungen zu ermöglichen.
- Das System erfordert ein rollenbasiertes Verhalten, das nachahmt, wie menschliche Teams (z. B. Projektmanager, Entwickler und Prüfer) zusammenarbeiten.

Die Orchestrierung ist ideal für Multiturn-Planungsagenten, Co-Piloten für die Softwareentwicklung, Prozessagenten im Unternehmen und autonome Projektausführer. Sie ist besonders nützlich bei der Implementierung von Systemen mit mehreren Agenten, die eine zentrale Aufschlüsselung der Aufgaben, aber eine verteilte Ausführungslogik erfordern, wodurch Erweiterbarkeit und ein besser erklärbares Verhalten auf allen Agentenebenen ermöglicht werden.

Capabilities

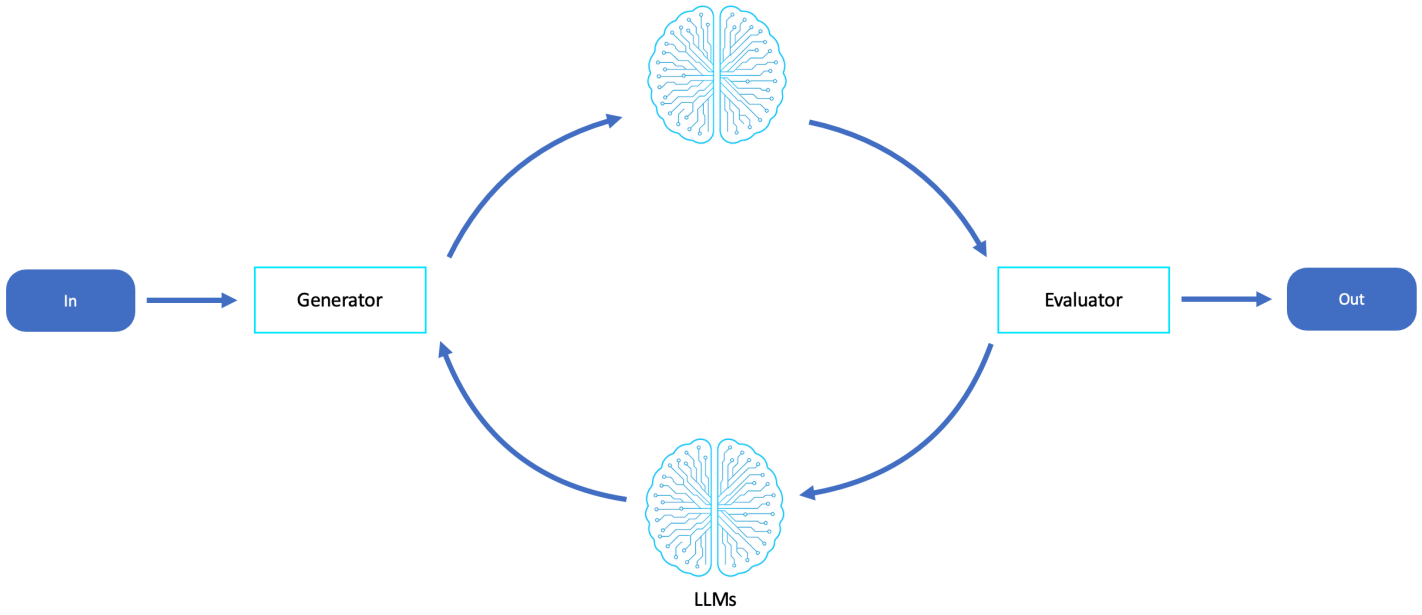
- Orchestrator führt Meta-Argumentation zur Zielsetzung durch
- Worker-Agents können Toolzugriff, Speicher- oder domänenspezifische Eingabeaufforderungen beinhalten
- Kann hierarchisch sein (d. h. mehrstufige Aufgabendelegierung)

Häufige Anwendungsfälle

- Projektmanager, koordinierende Forscher, Autoren und Qualitätssicherungsbeauftragte
- Codierungs-Copiloten, die Planung, Ausführung und Testen kombinieren
- Agenten, die Toolchains oder API-Zugriffsmuster überwachen

Arbeitsablauf für Evaluatoren und Reflect-Refine-Schleifen

Dieser Workflow bietet eine Feedback-Schleife, in der ein LLM ein Ergebnis generiert und ein anderer das Ergebnis bewertet oder kritisiert. Dies fördert Selbstreflexion, Optimierung und iterative Verbesserungen.



Der Arbeitsablauf für Evaluatoren ist ideal für Szenarien, in denen Qualität, Genauigkeit und Ausrichtung der Ergebnisse wichtig sind und in denen die Generierung in einem Durchgang unzuverlässig oder unzureichend ist. Dieser Workflow eignet sich hervorragend, wenn Agenten ihre Ergebnisse selbst kritisieren, iterieren und verfeinern müssen — entweder, um einen höheren Korrektheitsstandard zu erreichen oder um auf der Grundlage von Feedback verbesserte Alternativen zu untersuchen.

Dieser Workflow ist besonders effektiv, wenn:

- Das Ergebnis umfasst subjektive Qualitätskennzahlen (z. B. Stil, Ton und Lesbarkeit) oder objektive Kriterien (z. B. Richtigkeit, Sicherheit und Leistung).
- Der Mitarbeiter muss Kompromisse abwägen, Einschränkungen abwägen oder auf ein Ziel hin optimieren.
- Sie benötigen integrierte Redundanz und Qualitätssicherung, insbesondere in regulierten, kundenorientierten oder kreativen Bereichen.
- Human-in-the-loop Eine Überprüfung ist teuer oder nicht verfügbar, und eine unabhängige Validierung ist erwünscht.

Dieser Workflow wird für die Generierung von Inhalten, die Codesynthese und -überprüfung, die Durchsetzung von Richtlinien, die Überprüfung der Ausrichtung, die Anpassung von Anweisungen und die RAG-Nachbearbeitung verwendet. Er eignet sich auch für Mitarbeiter, die sich selbst

verbessern, da kontinuierliches Feedback dazu beiträgt, im Laufe der Zeit bessere Antworten zu entwickeln, um vertrauenswürdige, autonome Entscheidungsschleifen aufzubauen.

Häufige Anwendungsfälle

- Red-team Agenten im Vergleich zu Agenten mit blauem Team
- Agenten, die Code oder Pläne generieren, auswerten und überarbeiten
- Qualitätssicherung, Erkennung von Halluzinationen und Durchsetzung von Stilen

Capabilities

- Unterstützt die entkoppelte Generierung und Auswertung mithilfe verschiedener Modelle (z. B. Claude für die Generierung und Mistral für die Bewertung)
- Feedback wird strukturiert und verwendet, um zu überarbeiteten Ergebnissen zu gelangen
- Unterstützt mehrere Iterationen oder Konvergenzschwellenwerte

Schlussfolgerung

LLMs bilden den kognitiven Kern moderner Softwareagenten, aber der Aufruf von Rohmodellen reicht nicht aus, um zielgerichtete, robuste und kontrollierbare Intelligenz zu erreichen. Um von der Generierung von Output zu strukturiertem Denken und zielgerichtetem Verhalten überzugehen, müssen LLMs in gezielte Workflow-Muster eingebettet werden, die definieren, wie Modelle Eingaben verarbeiten, Kontexte verwalten und Aktionen koordinieren.

LLM-Workflows bieten Grundlagen für den Aufbau eines kognitiven Moduls für Agenten:

- Prompt Chaining unterteilt komplexes Denken in modulare, überprüfbare Schritte.
- Routing ermöglicht eine intelligente Aufgabenklassifizierung und gezielte Delegierung.
- Parallelisierung beschleunigt den Durchsatz und fördert vielfältige Argumentationsmöglichkeiten.
- Die Agentenorchestrierung strukturiert die Zusammenarbeit mehrerer Agenten durch Aufgabenzerlegung und rollenbasierte Ausführung.
- Der Evaluator (Reflect-Refine-Loop) ermöglicht Selbstverbesserung, Qualitätskontrolle und Überprüfung der Ausrichtung.

Jeder Workflow stellt ein zusammensetzbares Muster dar, das an die Bedürfnisse des Agenten, die Komplexität der Aufgabe und die Erwartungen des Benutzers angepasst werden kann. Diese Workflows schließen sich nicht gegenseitig aus. Sie sind Bausteine, die häufig zu Hybridarchitekturen kombiniert werden, die dynamisches Denken, Koordination mehrerer Agenten und Zuverlässigkeit auf Unternehmensniveau unterstützen.

Wenn Sie zum nächsten Kapitel über behördliche Workflow-Muster übergehen, werden diese LLM-Workflows wieder als eingebettete Strukturen in größeren Systemen auftauchen und die Delegierung von Zielen, die Orchestrierung von Tools, Entscheidungsschleifen und die Autonomie des Lebenszyklus unterstützen. Die Beherrschung dieser LLM-Workflows ist entscheidend für die Entwicklung von Softwareagenten, die nicht nur Text vorhersagen, sondern auch vernünftig argumentieren, sich anpassen und zielgerichtet handeln.

Agentäre Workflow-Muster

Agentic Workflow Patterns integrieren modulare Softwareagenten mit strukturierten LLM-Workflows (Large Language Model) und ermöglichen so autonomes Denken und Handeln. Diese Muster sind zwar von traditionellen serverlosen und ereignisgesteuerten Architekturen inspiriert, verlagern aber die Kernlogik von statischem Code auf LLM-augmented Agenten und bieten so eine verbesserte Anpassungsfähigkeit und kontextuelle Entscheidungsfindung. Diese Entwicklung transformiert konventionelle Cloud-Architekturen von deterministischen Systemen hin zu Systemen, die dynamische Interpretation und intelligente Erweiterung ermöglichen und gleichzeitig die Grundprinzipien der Skalierbarkeit und Reaktionsfähigkeit beibehalten.

In diesem Abschnitt

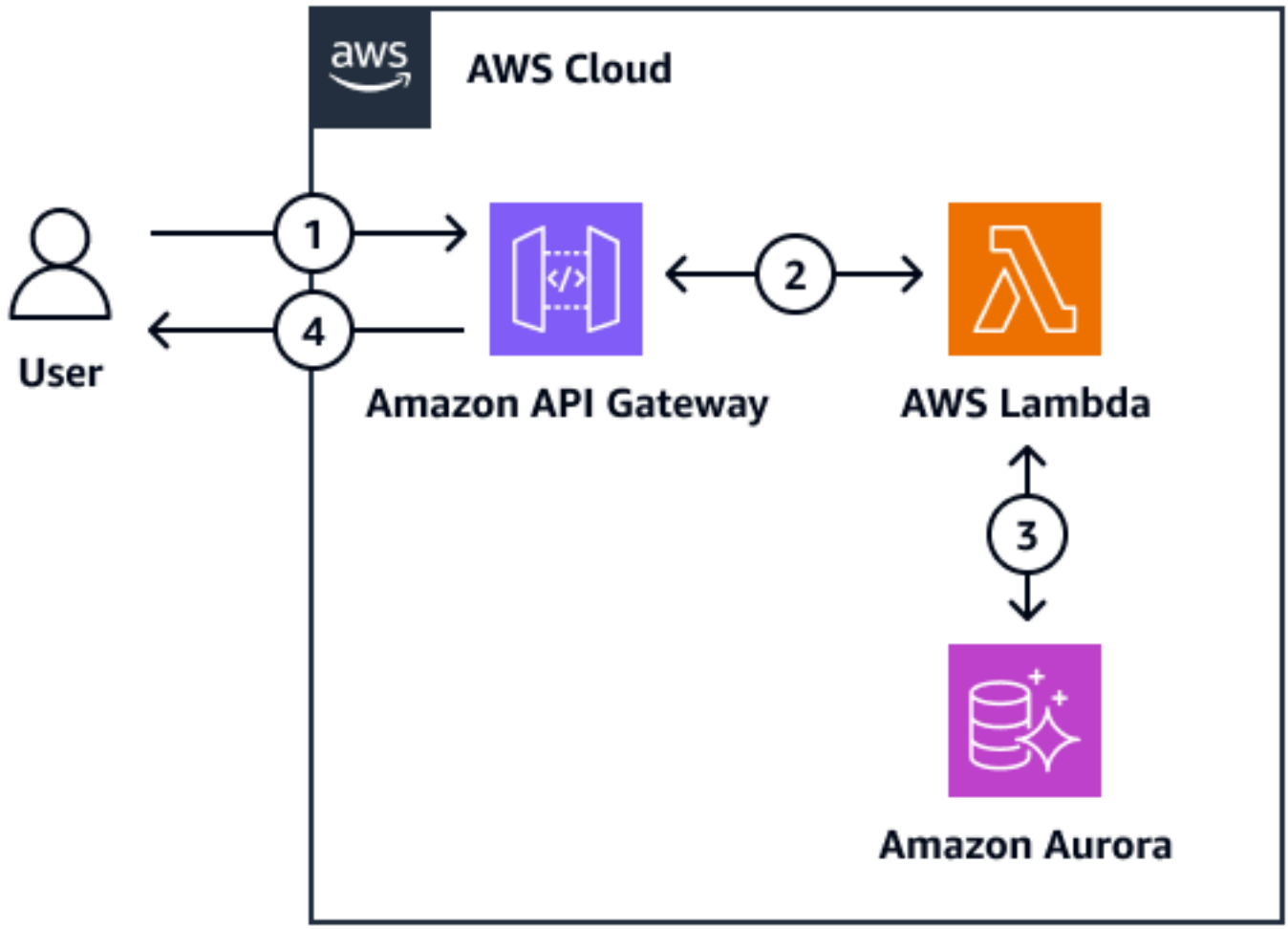
- Von ereignisgesteuerten zu kognitionserweiterten Systemen
- Sofortige Verkettung von Saga-Mustern
- Weiterleitung dynamischer Versandmuster
- Parallelisierung und Scatter-Gather-Muster
- Orchestrierungsmuster von Saga
- Schleifenmuster im Evaluator reflektieren/verfeinern
- Gestaltung agentischer Workflows auf AWS
- Schlussfolgerung

Von ereignisgesteuerten zu kognitionserweiterten Systemen

Moderne Cloud-Architekturen, insbesondere solche, die auf serverlosen und ereignisgesteuerten Prinzipien basieren, verlassen sich traditionell auf Muster wie Routing, Fan-Out und Anreicherung, um reaktionsschnelle, skalierbare Systeme zu schaffen. Agentische KI-Systeme bauen auf diesen Grundlagen auf und orientieren sie gleichzeitig an Argumentation und kognitiver Flexibilität. LLM-augmented Dieser Ansatz ermöglicht ausgefeiltere Problemlösungs- und Automatisierungsfunktionen und könnte die Art und Weise, wie komplexe Aufgaben in Cloud-Umgebungen gehandhabt werden, revolutionieren.

Event-driven Architektur

Das folgende Diagramm zeigt ein typisches verteiltes System:

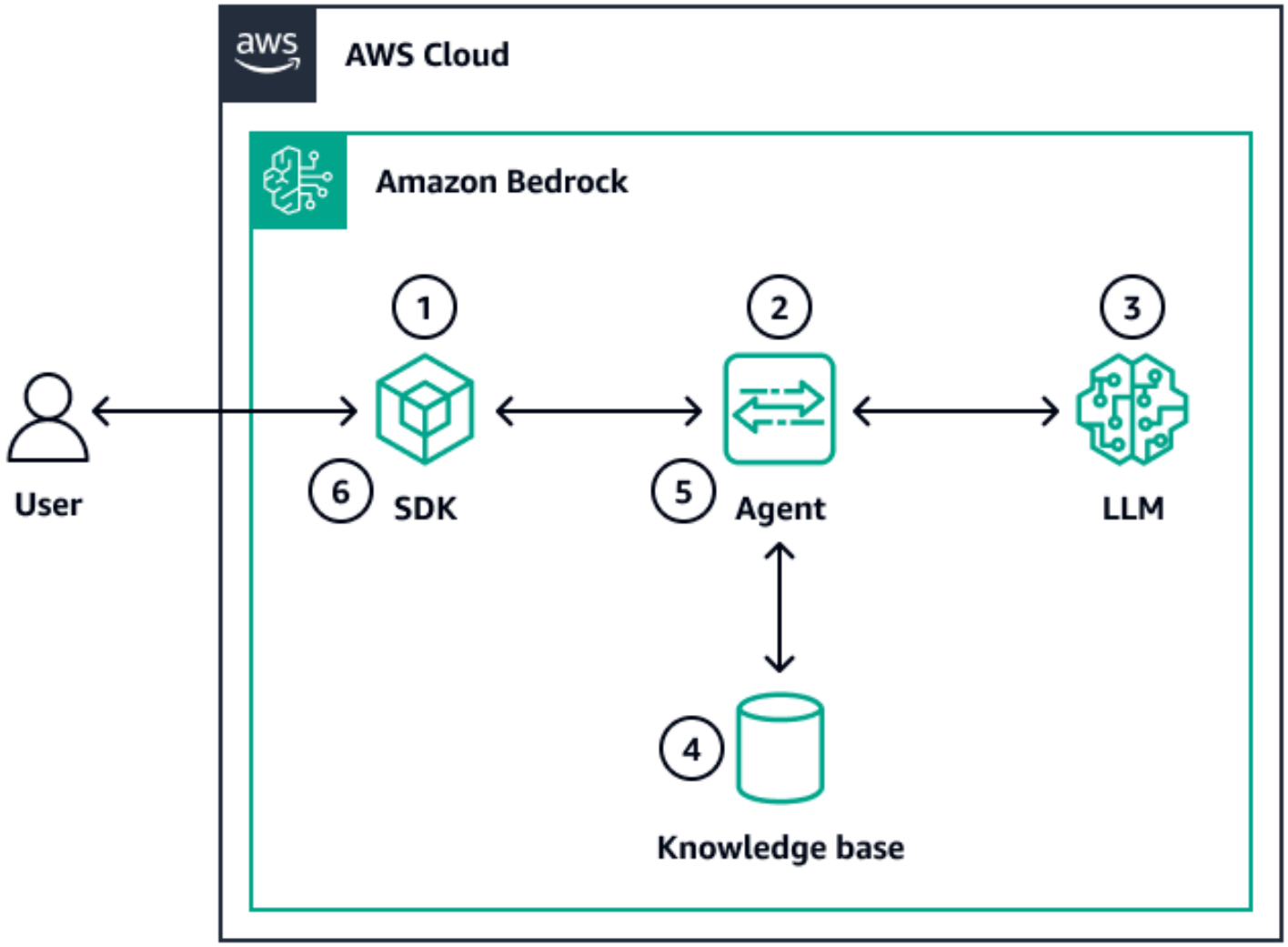


1. Ein Benutzer sendet eine Anfrage an Amazon API Gateway.
2. Amazon API Gateway leitet die Anfrage an eine AWS Lambda Funktion weiter.
3. AWS Lambda führt eine Datenanreicherung durch Abfrage einer Amazon Aurora Aurora-Datenbank durch
4. Amazon API Gateway gibt die angereicherte Nutzlast an den Aufrufer zurück.

Diese Struktur ist sowohl zuverlässig als auch skalierbar, aber sie ist grundsätzlich statisch. Geschäftsregeln und Logikpfade müssen explizit codiert werden, und die Anpassung an sich ändernde Kontexte oder unvollständige Informationen ist begrenzt.

Cognition-augmented Arbeitsabläufe

Agentenarchitekturen erweitern ein ereignisgesteuertes System um kognitive Erweiterungen. Das folgende Diagramm zeigt ein agentisches Äquivalent:



1. Ein Benutzer sendet eine Anfrage über einen SDK- oder API-Aufruf.
2. Ein Amazon Bedrock-Agent erhält die Anfrage.
3. Der Agent interpretiert die Anfrage, indem er ein LLM aufruft
4. Der Agent führt eine semantische Anreicherung durch, indem er die Amazon Bedrock-Wissensdatenbank oder andere externe Datenquellen durchsucht.
5. Das LLM synthetisiert eine kontextreiche, zielgerichtete Antwort.
6. Das System gibt dem Benutzer eine synthetisierte Antwort zurück.

In diesem Ablauf verwendet das LLM Logik, versteht die Absicht, ruft den relevanten Kontext ab, kombiniert ihn und entscheidet dann, wie am besten reagiert werden soll. Dieses Muster spiegelt das traditionelle Anreicherungsmuster wider, bei dem Nachrichten mit externen Daten angereichert werden, bevor sie weitergeleitet werden. In Agentensystemen handelt es sich bei dieser

Anreicherung jedoch nicht um eine statische Suche. Stattdessen ist die Anreicherung dynamisch, semantisch gesteuert und zielgerichtet.

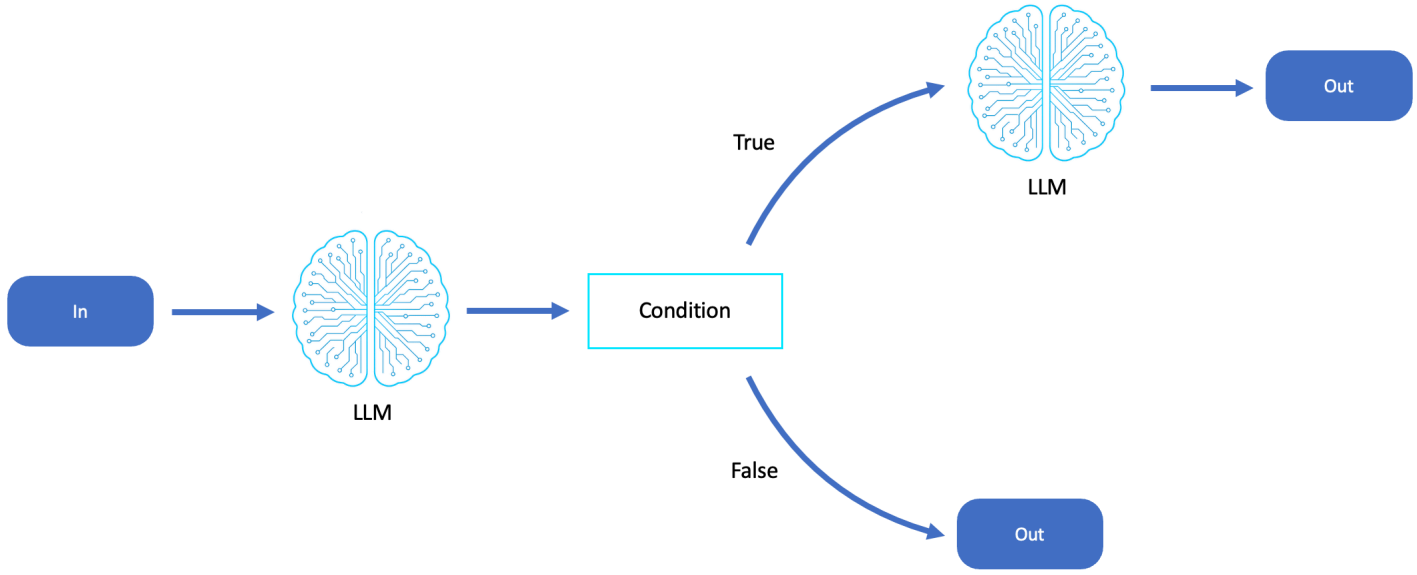
Die wichtigsten Erkenntnisse

Jeder LLM-Workflow kann einem agentischen Workflow-Muster zugeordnet werden, das traditionelle ereignisgesteuerte Architekturstile widerspiegelt und weiterentwickelt. Ein grundlegender Baustein agentischer Workflows ist die Fähigkeit, den Kontext eines LLM um Daten, Tools und Speicher zu erweitern. Dadurch entsteht eine Argumentationsschleife, die fundiert, anpassungsfähig und auf die Benutzerabsichten abgestimmt ist. Während herkömmliche Systeme Nachrichten mit Suchdaten anreichern, ermöglichen es Agentensysteme der Software, sich weniger wie Skripte, sondern eher wie intelligente Kollaborateure zu verhalten.

Sofortige Verkettung von Saga-Mustern

Indem wir LLM Prompt Chaining als ereignisgesteuerte Saga neu erfinden, erschließen wir ein neues Betriebsmodell: Workflows werden auf autonome Agenten verteilt, wiederherstellbar und semantisch koordiniert. Jeder Prompt-Response-Schritt wird als atomare Aufgabe neu formuliert, als Ereignis ausgegeben, von einem speziellen Agenten verarbeitet und mit kontextuellen Metadaten angereichert.

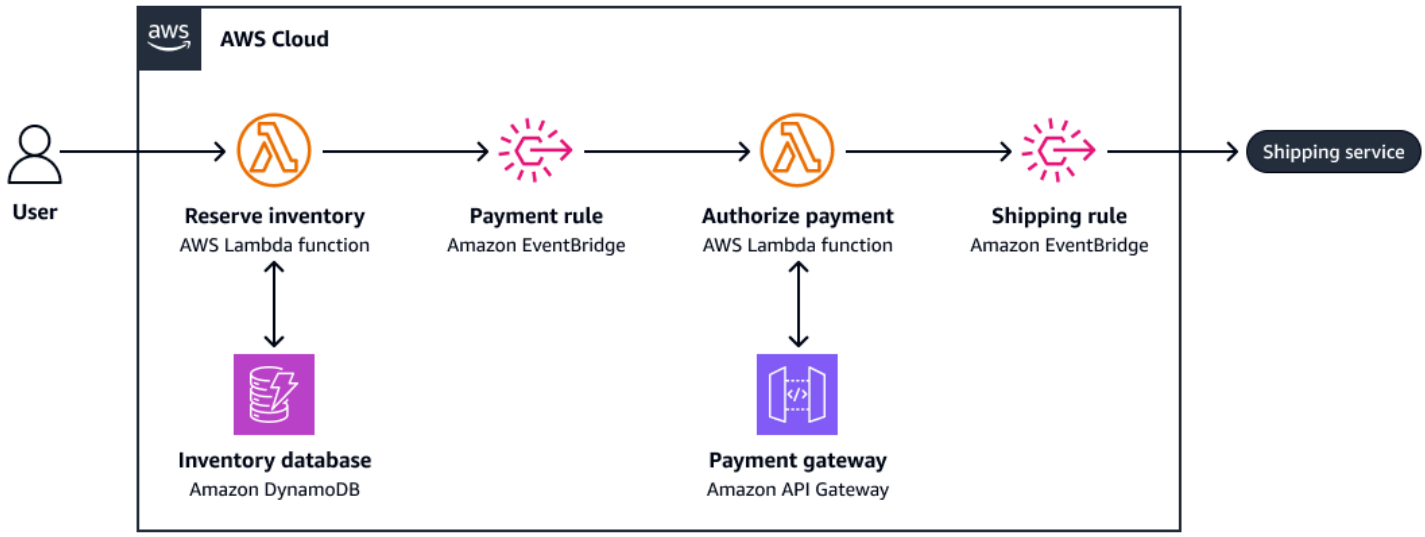
Das folgende Diagramm ist ein Beispiel für die Verkettung von LLM-Eingabeaufforderungen:



Saga-Choreographie

Das Saga-Choreographie-Muster ist ein Implementierungsansatz in verteilten Systemen, für den es keinen zentralen Koordinator gibt. Stattdessen veröffentlicht jeder Dienst oder jede Komponente Ereignisse, die die nächste Workflow-Aktion auslösen. Dieses Muster wird häufig in verteilten Systemen zur Verwaltung von Transaktionen über mehrere Dienste hinweg verwendet. In einer Saga führt das System eine Reihe koordinierter lokaler Transaktionen durch. Wenn eine davon ausfällt, löst das System Ausgleichsmaßnahmen aus, um die Konsistenz aufrechtzuerhalten.

Das folgende Diagramm ist ein Beispiel für eine Saga-Choreographie:



1. Inventar reservieren
2. Zahlung autorisieren
3. Versandauftrag erstellen

Schlägt Schritt 3 fehl, leitet das System Ausgleichsmaßnahmen ein (z. B. eine Zahlung stornieren oder Inventar freigeben).

Dieses Muster ist besonders nützlich in ereignisgesteuerten Architekturen, in denen Dienste lose miteinander verknüpft sind und Zustände im Laufe der Zeit konsistent behoben werden müssen, selbst bei teilweisem Ausfall.

Muster für die schnelle Verkettung

Die schnelle Verkettung ähnelt in ihrer Struktur und ihrem Zweck dem Saga-Muster. Es führt eine Reihe von Argumentationsschritten aus, die nacheinander aufgebaut werden, wobei der Kontext erhalten bleibt und Rollbacks und Überarbeitungen möglich sind.

Choreographie der Agenten

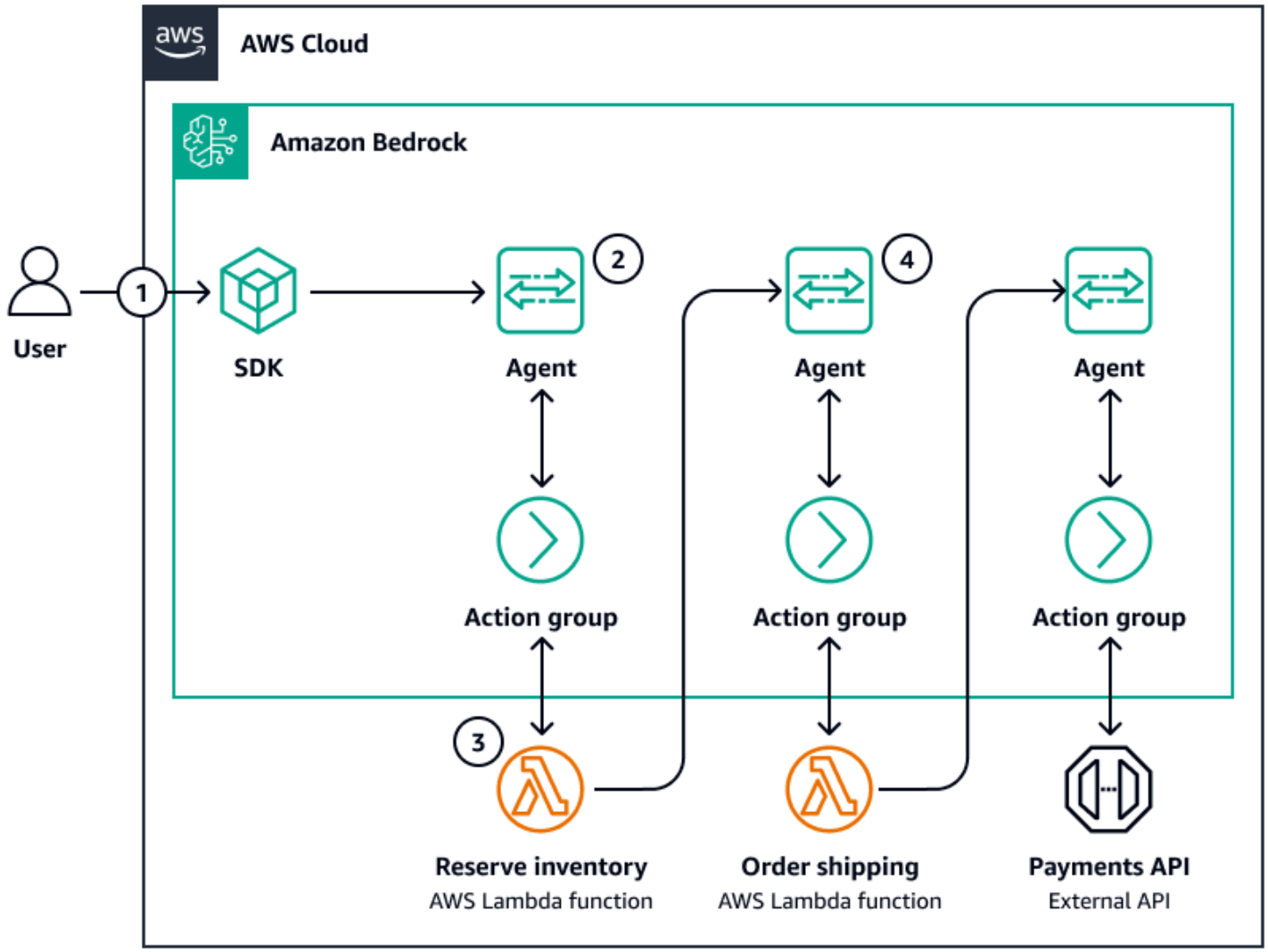
1. LLM interpretiert eine komplexe Benutzerabfrage und generiert eine Hypothese
2. LLM erarbeitet einen Plan zur Lösung der Aufgabe
3. LLM führt eine Unteraufgabe aus (z. B. mithilfe eines Tool-Aufrufs oder durch das Abrufen von Wissen)
4. LLM verfeinert die Ausgabe oder wiederholt einen früheren Schritt, wenn es ein Ergebnis für unbefriedigend hält

Wenn ein Zwischenergebnis fehlerhaft ist, kann das System einen der folgenden Schritte ausführen:

- Wiederholen Sie die Schritte mit einem anderen Ansatz
- Kehren Sie zu einer vorherigen Eingabeaufforderung zurück und planen Sie neu
- Verwenden Sie eine Evaluatorschleife (z. B. anhand des Evaluator-Optimizer-Musters), um Fehler zu erkennen und zu korrigieren

Wie beim Saga-Muster ermöglicht die Verkettung von Eingabeaufforderungen teilweise Fortschritts- und Rollback-Mechanismen. Dies geschieht eher durch iterative Verfeinerung und LLM-directed Korrektur als durch kompensierende Datenbanktransaktionen.

Das folgende Diagramm ist ein Beispiel für die Choreographie eines Agenten:



1. Ein Benutzer sendet eine Anfrage über ein SDK.
2. Ein Amazon Bedrock-Agent orchestriert die Argumentation wie folgt:
   - Dolmetschen (LLM)
   - Planung (LLM)
   - Ausführung über ein Tool oder eine Wissensdatenbank
   - Konstruktion der Antwort
3. Wenn ein Tool ausfällt oder unzureichende Daten zurückgibt, kann der Agent die Aufgabe dynamisch neu planen oder umformulieren.
4. Speicher (z. B. ein kurzfristiger Vektorspeicher) kann seinen Zustand stufenübergreifend beibehalten

Imbissbuden

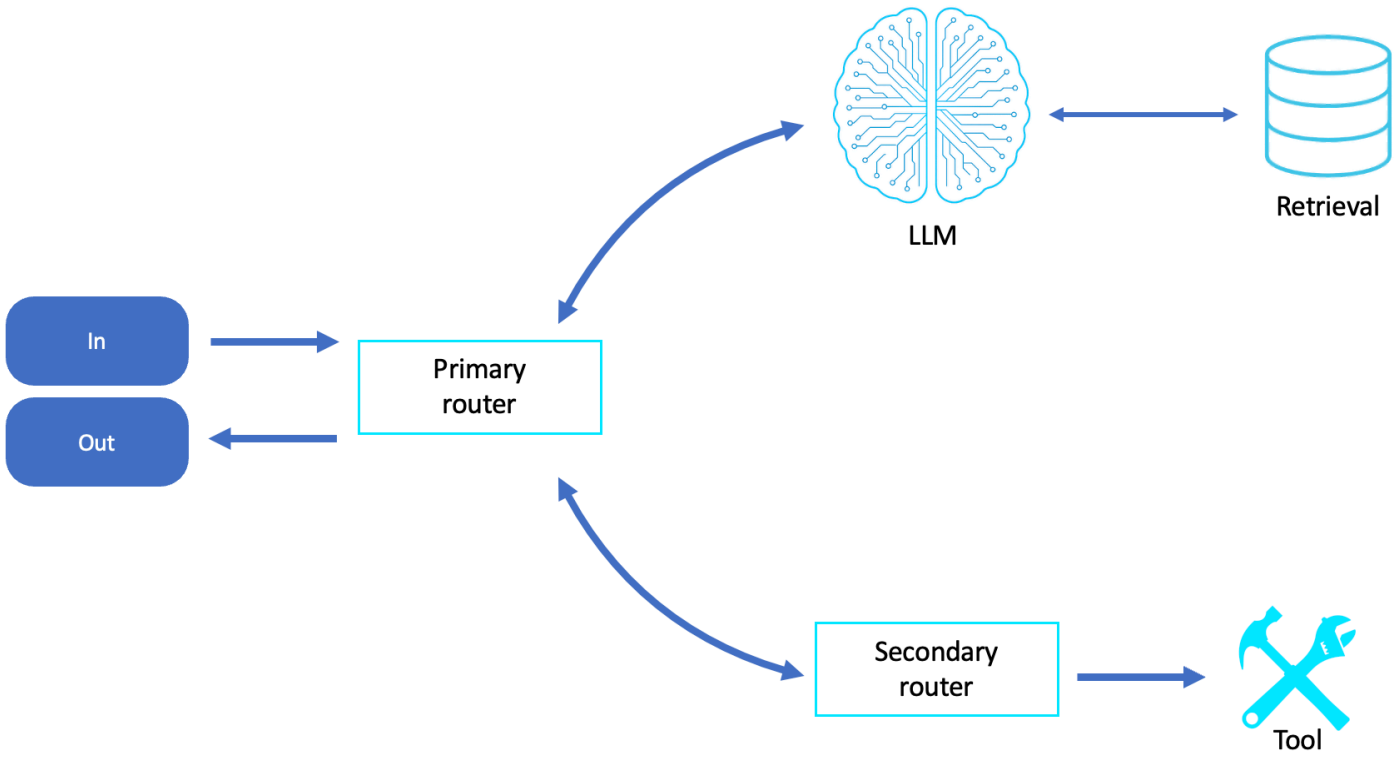
Während das Saga-Muster verteilte Serviceanrufe mit kompensierender Logik verwaltet, erledigt Prompt Chaining Denkaufgaben mit reflektierter Reihenfolge und adaptiver Neuplanung. Beide Systeme ermöglichen inkrementellen Fortschritt, dezentrale Entscheidungspunkte und die Behebung von Fehlern, und das alles durch fundierte Argumentation und nicht durch starres Rollback.

Prompt Chaining führt Transaktionsdenken ein, das kognitive Äquivalent von Sagen ist. Das heißt, jeder „Gedanke“ wird im Rahmen eines umfassenderen zielgerichteten Dialogs neu bewertet, überarbeitet oder aufgegeben.

Weiterleitung dynamischer Versandmuster

In modernen Agentensystemen, in denen die Aufgaben vom Analysieren von Dokumenten bis hin zur autonomen Softwaregenerierung reichen, ist die Fähigkeit, Anfragen dynamisch an das leistungsfähigste Large Language Model (LLM) oder den leistungsfähigsten Agenten weiterzuleiten, von entscheidender Bedeutung. Der statischen Routing-Logik, die häufig in Orchestrierungsskripten oder API-Ebenen eingebettet ist, fehlt die Anpassungsfähigkeit, die für Umgebungen mit mehreren Modellen und Funktionen in Echtzeit erforderlich ist. Um diesem Problem zu begegnen, können LLM-Routing-Workflows in eine ereignisgesteuerte Architektur umgewandelt werden, die ein dynamisches Versandmuster nutzt und LLM-Aufrufe in intelligent geroutete, kontextsensitive Ereignisse umwandelt.

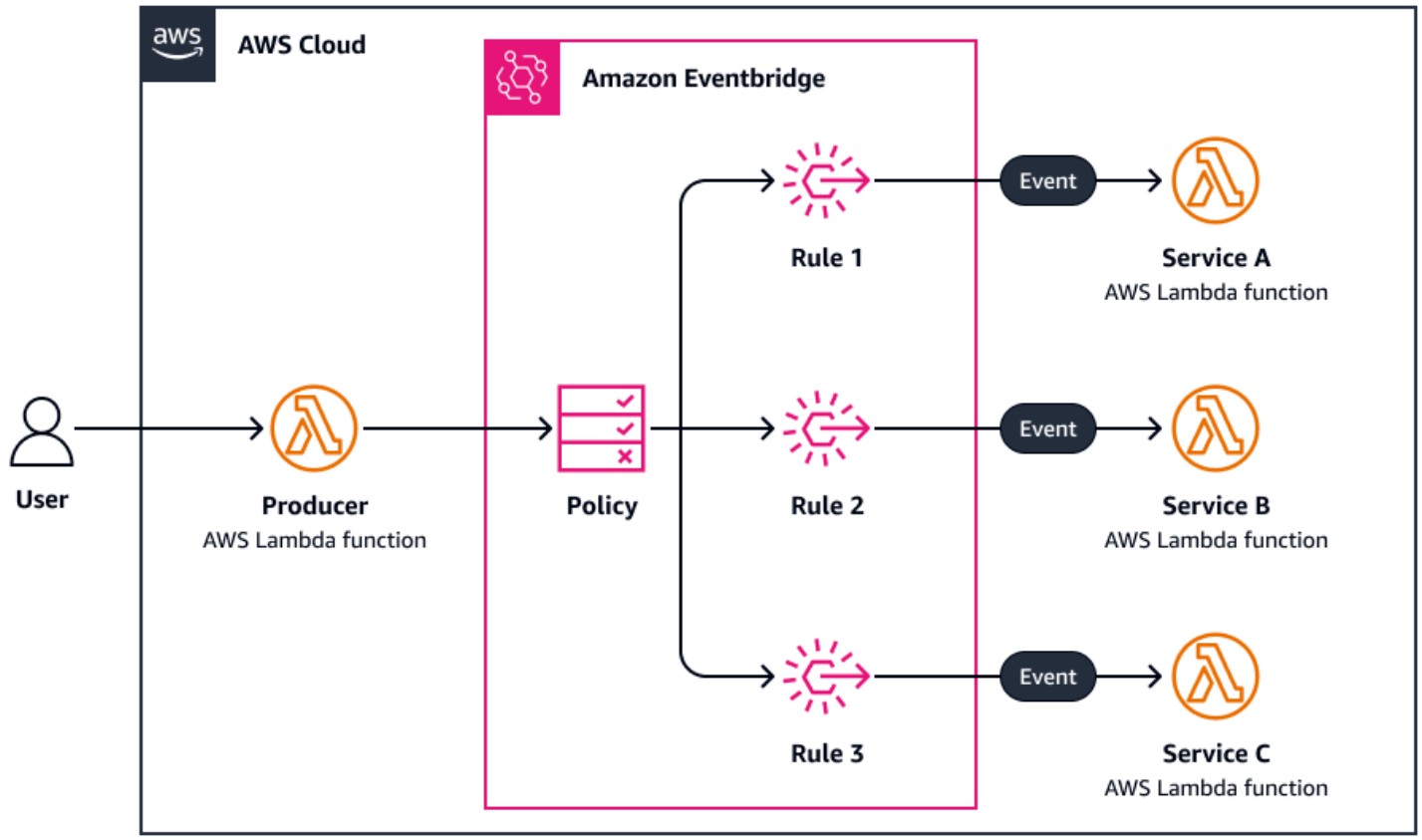
Das folgende Diagramm ist ein Beispiel für LLM-Routing:



Dynamischer Versand

In herkömmlichen verteilten Systemen wählt das dynamische Versandmuster bestimmte Dienste zur Laufzeit aus und ruft sie auf der Grundlage eingehender Ereignisattribute wie Ereignistyp, Quelle und Nutzlast aus. Dies wird üblicherweise mithilfe von Amazon implementiert EventBridge, das eingehende Ereignisse auswerten und an geeignete Ziele weiterleiten kann (z. AWS Step Functions B. AWS Lambda Funktionen oder Amazon Elastic Container Service-Aufgaben).

Das folgende Diagramm ist ein Beispiel für dynamischen Versand:



1. Eine Anwendung gibt ein Ereignis aus (z. B. {"type“: „orderCreated“, „priority“: „high"}).
2. Amazon EventBridge bewertet das Ereignis anhand seiner Routing-Regeln.
3. Basierend auf den Attributen eines Ereignisses sendet das System dynamisch an Folgendes:
   - `HighPriorityOrderProcessor`(Dienst A)
   - `StandardOrderProcessor`(Dienst B)
   - `UpdateOrderProcessor`(Dienst C)

Dieses Muster unterstützt lose Kopplung, domänenbasierte Spezialisierung und Laufzeiterweiterbarkeit. Dadurch können Systeme intelligent auf sich ändernde Anforderungen und die Semantik von Ereignissen reagieren.

LLM-based Routing

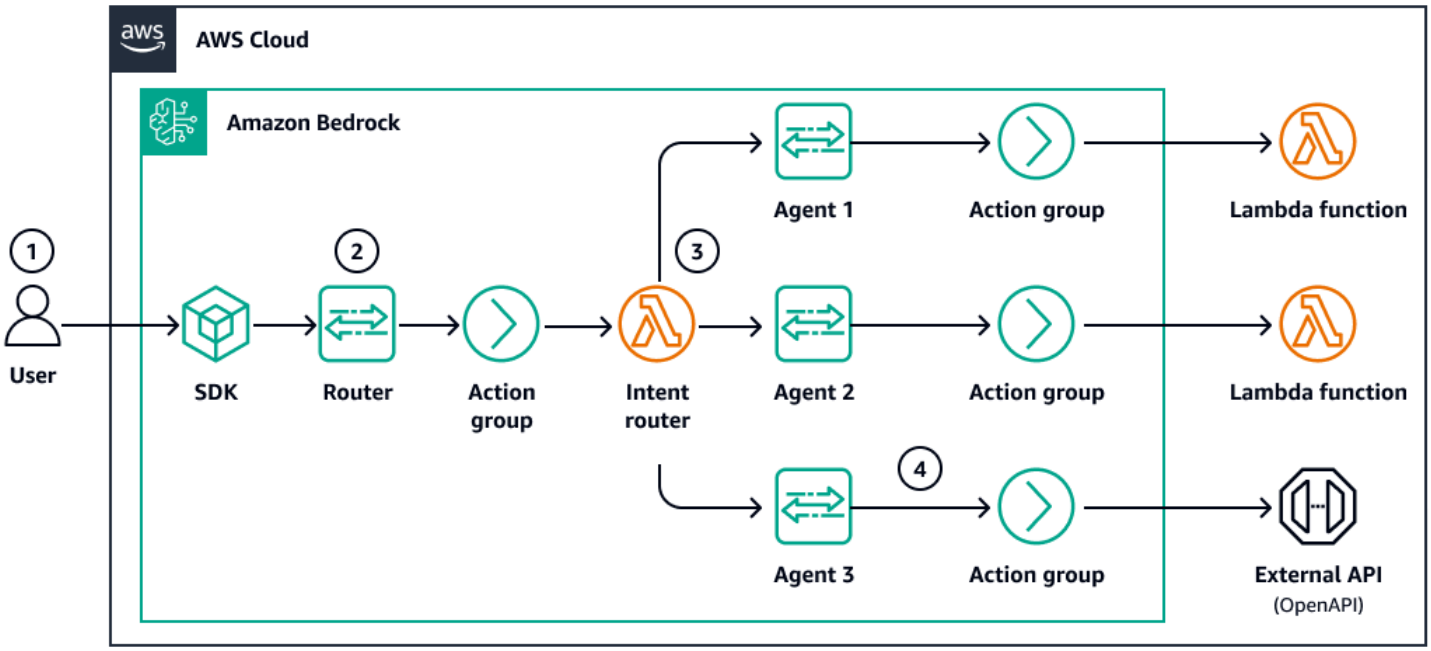
In Agentensystemen führt das Routing auch eine dynamische Aufgabendelegierung durch — aber anstelle von EventBridge Amazon-Regeln oder Metadatenfiltern klassifiziert und interpretiert das LLM die Absicht des Benutzers in natürlicher Sprache. Das Ergebnis ist eine flexible, semantische und anpassungsfähige Form des Dispatchings.

Agenten-Router

Diese Architektur ermöglicht ein funktionsorientiertes Dispatching ohne vordefinierte Schemas oder Ereignistypen, was sich ideal für unstrukturierte Eingaben und komplexe Abfragen eignet.

1. Ein Benutzer sendet die Anfrage „Können Sie mir helfen, meine Vertragsbedingungen zu überprüfen?“
2. Das LLM interpretiert dies als eine Aufgabe für ein rechtliches Dokument.
3. Der Agent leitet die Aufgabe an eine oder mehrere der folgenden Stellen weiter:
   - Vorlage für die Aufforderung zur Vertragsprüfung
   - Subagent für rechtliche Überlegungen
   - Tool zum Analysieren von Dokumenten

Das folgende Diagramm ist ein Beispiel für einen Agent-Router:



1. Ein Benutzer sendet eine Anfrage in natürlicher Sprache über ein SDK.
2. Ein Amazon Bedrock-Agent verwendet ein LLM, um die Aufgabe zu klassifizieren (z. B. rechtlich, technisch oder terminlich).
3. Der Agent leitet die Aufgabe dynamisch über eine Aktionsgruppe weiter, um den erforderlichen Agenten aufzurufen:
   - Domain-specific Agent

- Spezialisierte Werkzeugkette
- Benutzerdefinierte Prompt-Konfiguration

4. Der ausgewählte Handler verarbeitet die Aufgabe und gibt eine maßgeschneiderte Antwort zurück.

Imbissbuden

Während herkömmlicher dynamischer Versand EventBridge Amazon-Regeln für die Weiterleitung auf der Grundlage strukturierter Ereignisattribute verwendet, verwendet agentisches Routing LLMs, um Aufgaben anhand von Bedeutung und Absicht semantisch zu klassifizieren und weiterzuleiten. Dies erweitert die Flexibilität des Systems, indem es Folgendes ermöglicht:

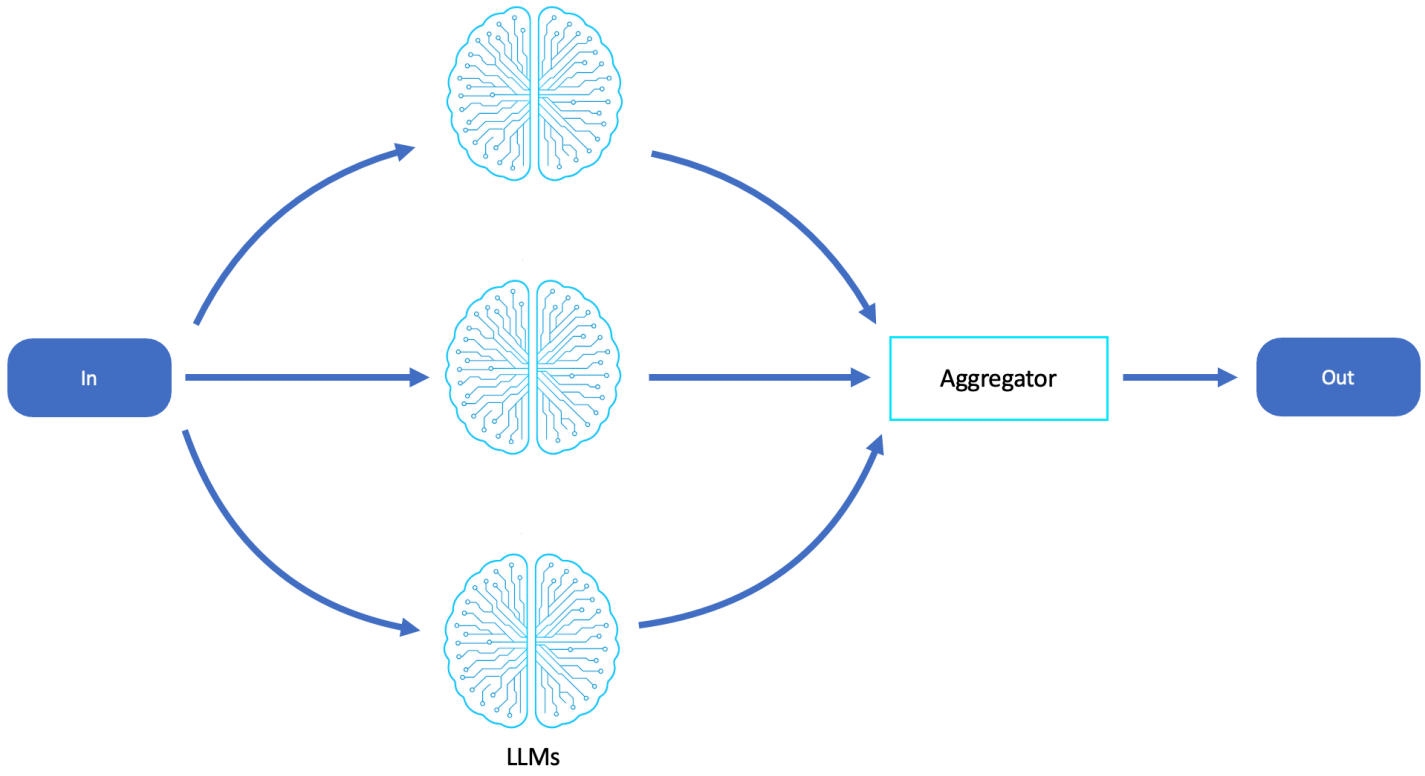
- Umfassenderes Verständnis der
- Intelligenter Fallback und Toolauswahl
- Natürliche Erweiterbarkeit durch neue Agentenrollen oder Eingabeaufforderungsstile

Agentic Routing ersetzt starre Regeln durch dynamisches kognitives Dispatching, das es Systemen ermöglicht, sich mit Sprache und nicht mit Code weiterzuentwickeln.

Parallelisierung und Scatter-Gather-Muster

Viele fortgeschrittene Denk- und Generierungsaufgaben — wie das Zusammenfassen großer Dokumente, die Bewertung mehrerer Lösungswege oder der Vergleich verschiedener Perspektiven — profitieren von der parallel Ausführung von Eingabeaufforderungen. Herkömmliche sequentielle Workflows reichen nicht aus, wenn Skalierbarkeit, Reaktionsfähigkeit und Fehlertoleranz erforderlich sind. Um dieses Problem zu umgehen, kann LLM-based Parallelisierung mithilfe eines ereignisgesteuerten Scatter-Gather-Musters neu konzipiert werden, bei dem Aufgaben dynamisch an autonome Agenten verteilt und die Ergebnisse intelligent synthetisiert werden.

Das folgende Diagramm ist ein Beispiel für einen LLM-Parallelisierungs-Workflow:



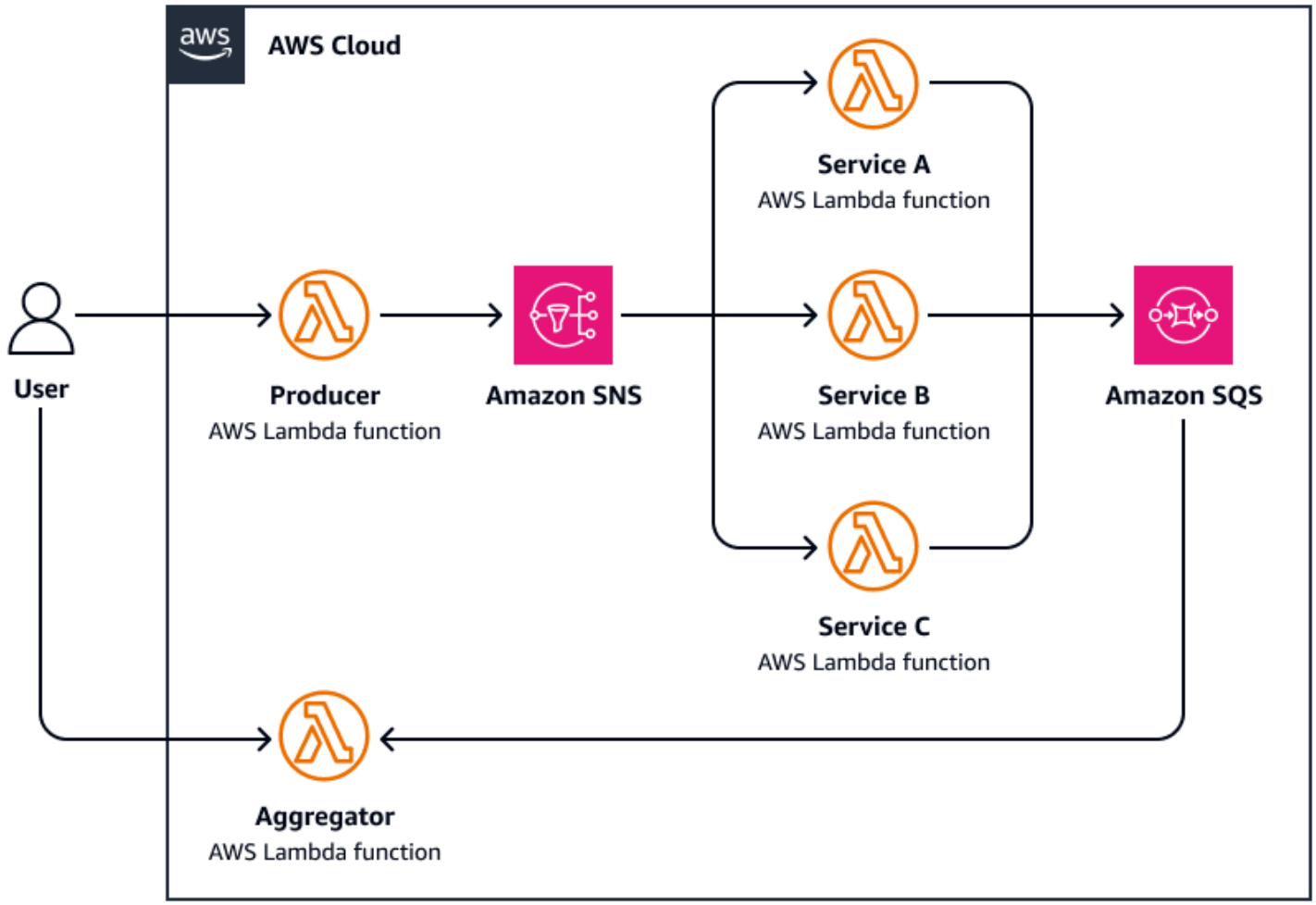
Scatter-gather

In verteilten Systemen sendet ein Scatter-Gather-Muster Aufgaben parallel an mehrere Dienste oder Verarbeitungseinheiten, wartet auf ihre Antworten und aggregiert dann die Ergebnisse zu einer konsolidierten Ausgabe. Im Gegensatz zu Fan-Out wird Scatter-Gathering koordiniert, weil es Antworten erwartet und in der Regel Logik anwendet, um Ergebnisse zu kombinieren, zu vergleichen und auszuwählen.

Zu den gängigen Implementierungen für Parallelisierung und Scatter-Gather gehören die folgenden:

- AWS Step Functions einen Status für die parallel Aufgabenausführung zuordnen
- AWS Lambda mit Parallelität, Koordination der Ergebnisse mehrerer aufgerufener Funktionen
- Amazon EventBridge mit Korrelations-IDs und Aggregationsworkflows
- Benutzerdefiniertes Controller-Muster zur Verwaltung von Fan-Outs und zur Erfassung von Ergebnissen mithilfe von Amazon Simple Storage Service (Amazon S3), Amazon DynamoDB oder Warteschlangen

Das folgende Diagramm ist ein Beispiel für Scatter-Gather:



1. Ein Benutzer sendet eine Anfrage an eine zentrale Koordinatorfunktion, die die Aufgabe verteilt, indem sie parallel Nachrichten zu einem Amazon Simple Notification Service (Amazon SNS) - Thema veröffentlicht.
2. Jede Nachricht enthält Aufgabenmetadaten und wird an einen Facharbeiter weitergeleitet. AWS Lambda
3. Jeder Worker verarbeitet AWS Lambda unabhängig die ihm zugewiesene Unteraufgabe (z. B. das Abfragen einer externen API, das Verarbeiten eines Dokuments und das Analysieren von Daten).
4. Die Ergebnisse werden auf eine gemeinsame Speicherebene wie Amazon Simple Queue Service (Amazon SQS) geschrieben.
5. Die Aggregatorfunktion wartet, bis alle Antworten abgeschlossen sind, und führt dann Folgendes aus:
   - Sammelt und aggregiert die Ergebnisse (führt beispielsweise Zusammenfassungen zusammen und wählt die besten Treffer aus)

- Sendet eine endgültige Antwort oder löst einen nachgelagerten Workflow aus

Zu den häufigsten Anwendungsfällen für Scatter-Gather-Muster gehören:

- Föderierte Suche
- Suchmaschinen für Preisvergleiche
- Aggregierte Datenanalyse
- Inferenz mit mehreren Modellen

LLM-based Parallelisierung (Scatter-Gather-Kognition)

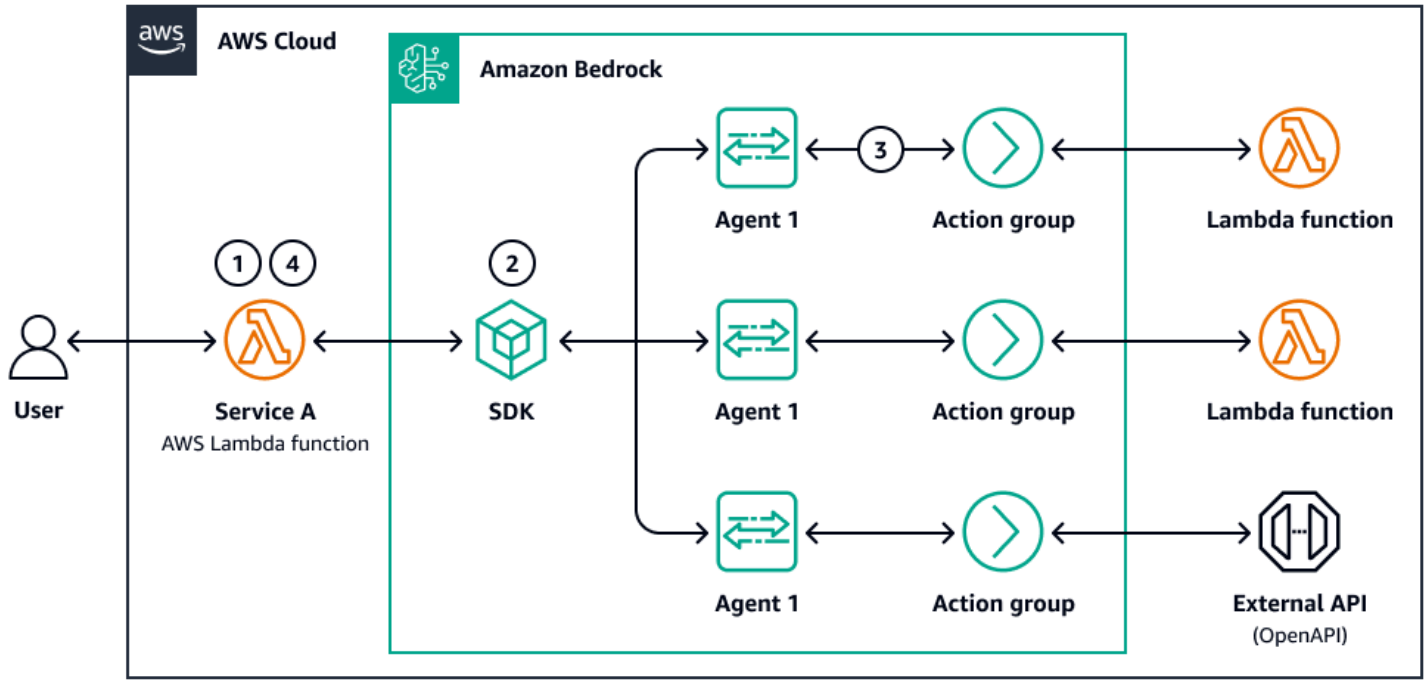
In Agentensystemen spiegelt die Parallelisierung stark das Scatter-Gather-Verfahren wider, indem Teilaufgaben auf mehrere LLM-Aufrufe oder -Agenten verteilt werden, die jeweils unabhängig voneinander einen Teil des Problems lösen. Die zurückgegebenen Ergebnisse werden durch einen Aggregationsprozess gesammelt und synthetisiert, bei dem es sich häufig um einen anderen LLM- oder Controller-Agenten handelt.

Parallelisierung der Agenten

1. Ein Agent reicht die Anfrage „Zusammenfassung der Erkenntnisse aus diesen 10 Berichten“ ein.
2. Es verteilt die Berichte auf 10 parallel LLM-Zusammenfassungsaufgaben.
3. Wenn er alle Zusammenfassungen zurückgibt, geht der Agent wie folgt vor:
   - Fasst Zusammenfassungen zu einem einheitlichen Briefing zusammen
   - Identifiziert Themen oder Widersprüche
   - Sendet die synthetisierte Ausgabe an den Benutzer

Dieser agentische Workflow ermöglicht skalierbares, modulares und adaptives paralleles Denken. Dies ist ideal für Anwendungsfälle, die einen hohen kognitiven Durchsatz erfordern.

Das folgende Diagramm ist ein Beispiel für die Parallelisierung von Agenten:



1. Ein Benutzer reicht eine mehrteilige Abfrage oder einen Dokumentensatz ein.
2. Eine Controller AWS Lambda - oder Schrittfunktion verteilt die Unteraufgaben. Jede Aufgabe ruft einen Amazon Bedrock LLM-Aufruf oder -Subagent mit eigener Aufforderung auf.
3. Wenn die Aufrufe und Unteraufgaben abgeschlossen sind, werden die Ergebnisse gespeichert (z. B. in Amazon S3 oder im Speicherspeicher), und ein Aggregationsschritt führt die Ausgaben zusammen, vergleicht oder filtert sie.
4. Das System sendet die endgültige Antwort an den Benutzer oder den nachgeschalteten Agenten zurück.

Dieses System verfügt über eine verteilte Argumentationsschleife mit Rückverfolgbarkeit, Fehlertoleranz und optionaler Ergebnisgewichtungs- oder Auswahllogik.

Imbissbuden

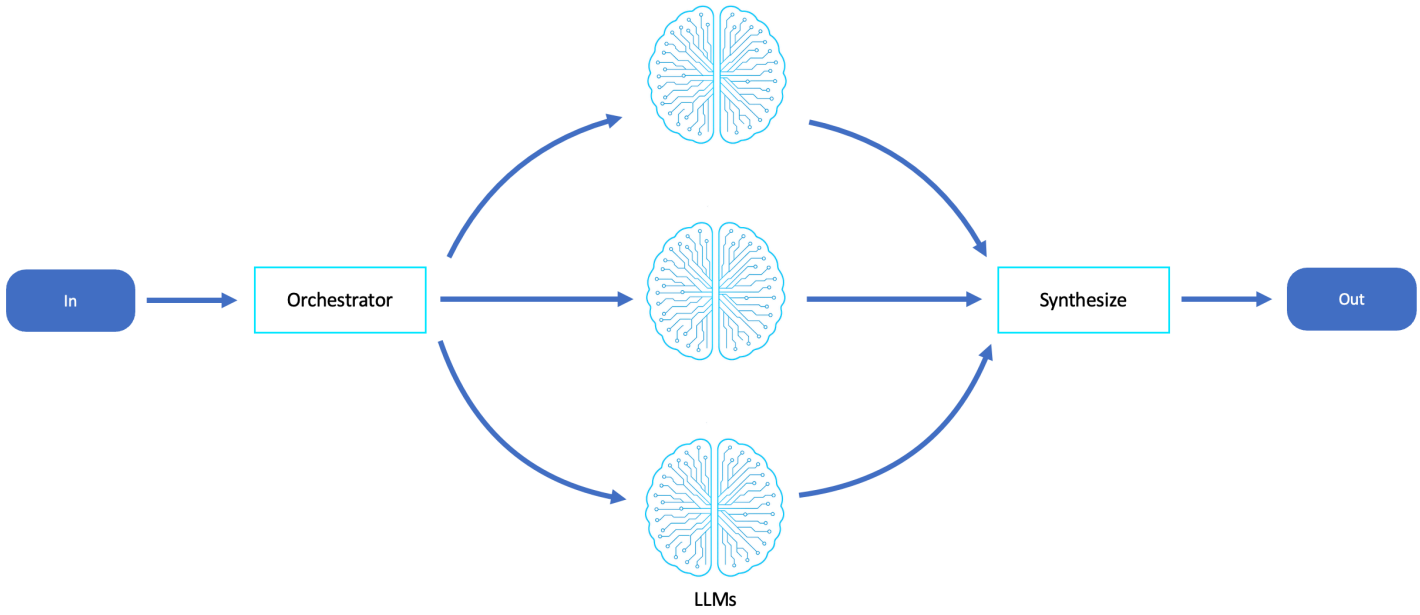
Die Agentenparallelisierung verwendet Scatter-Gather-Muster, um LLM-Aufgaben zu verteilen, was eine parallel Verarbeitung und eine intelligente Ergebnissynthese ermöglicht.

Orchestrierungsmuster von Saga

Da die von LLMs gesteuerten Workflows immer komplexer werden und Eingabeaufforderungsketten, Datenverarbeitungsschritte, Toolaufrufen und Zusammenarbeit mit Agenten umfassen, wird die

Notwendigkeit einer intelligenten Orchestrierung immer wichtiger. Anstatt sich auf eng gekoppelte Skripte oder statische, vorgegebene Ausführungsabläufe zu verlassen, können diese Workflows als ereignisgesteuerte Orchestrierungsmuster implementiert werden, sodass LLM-based Systeme mehrstufige Aufgaben zwischen autonomen Agenten dynamisch koordinieren, überwachen und anpassen können.

Das folgende Diagramm ist ein Beispiel für einen Orchestrator:



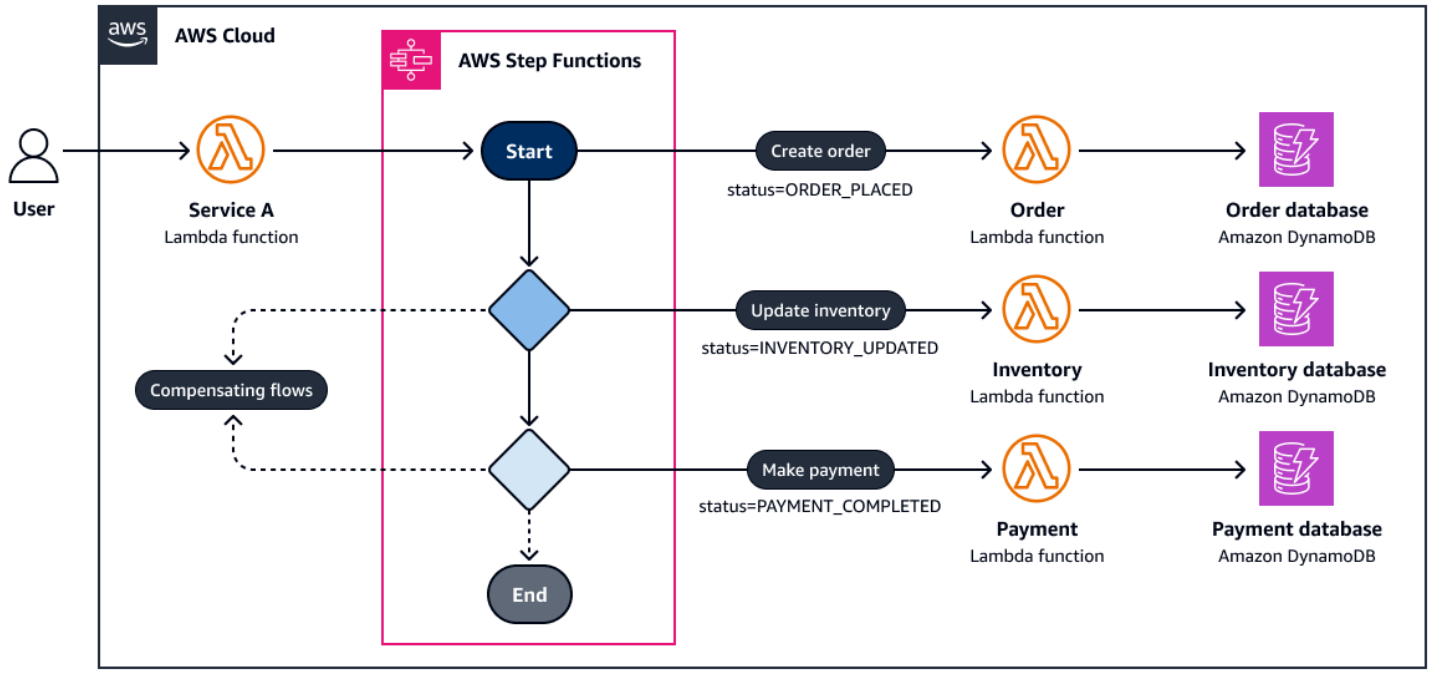
Orchestrierung von Veranstaltungen

In herkömmlichen verteilten Systemen bezieht sich Event Orchestration auf ein Muster, bei dem ein zentraler Koordinator einen komplexen Arbeitsablauf verwaltet, indem er den Kontrollfluss explizit über mehrere Dienste oder Aufgaben hinweg lenkt. Im Gegensatz zur Event-Choreographie (bei der jeder Service unabhängig reagiert) bietet die Orchestrierung eine zentralisierte Logik, Transparenz und Kontrolle über den gesamten Prozess.

Dies wird in der Regel mit den folgenden Tools implementiert:

- AWS Step Functions— Stateful-Workflows definieren und ausführen
- AWS Lambda— Führen Sie diskrete Aufgaben innerhalb des orchestrierten Ablaufs aus
- Amazon SQS oder Amazon EventBridge — Löst asynchrone Schritte oder Antworten aus

Das folgende Diagramm ist ein Beispiel für die Saga-Orchestrierung:



Ein AWS Step Functions Workflow verwaltet einen Kundenbestellvorgang:

1. Bestellung erstellen (AWS Lambda)
2. Inventar aktualisieren (AWS Lambda)
3. Zahlung tätigen (AWS Lambda)

Der Orchestrator koordiniert jeden Schritt, indem er Wiederholungen, parallel Verzweigungen, Timeouts und Fehler verwaltet.

Role-based Agentensystem (Orchestrator)

In Agentensystemen spiegelt das Orchestrator-Muster die Orchestrierung von Ereignissen wider, verteilt die Logik jedoch auf mehrere Argumentationsagenten, von denen jeder eine definierte Rolle oder Spezialisierung hat. Ein zentraler Orchestrator-Agent interpretiert die Gesamtaufgabe, zerlegt sie in Unteraufgaben und delegiert diese an Worker Agents, die jeweils für einen bestimmten Bereich optimiert sind (z. B. Recherche, Programmierung, Zusammenfassung, Überprüfung).

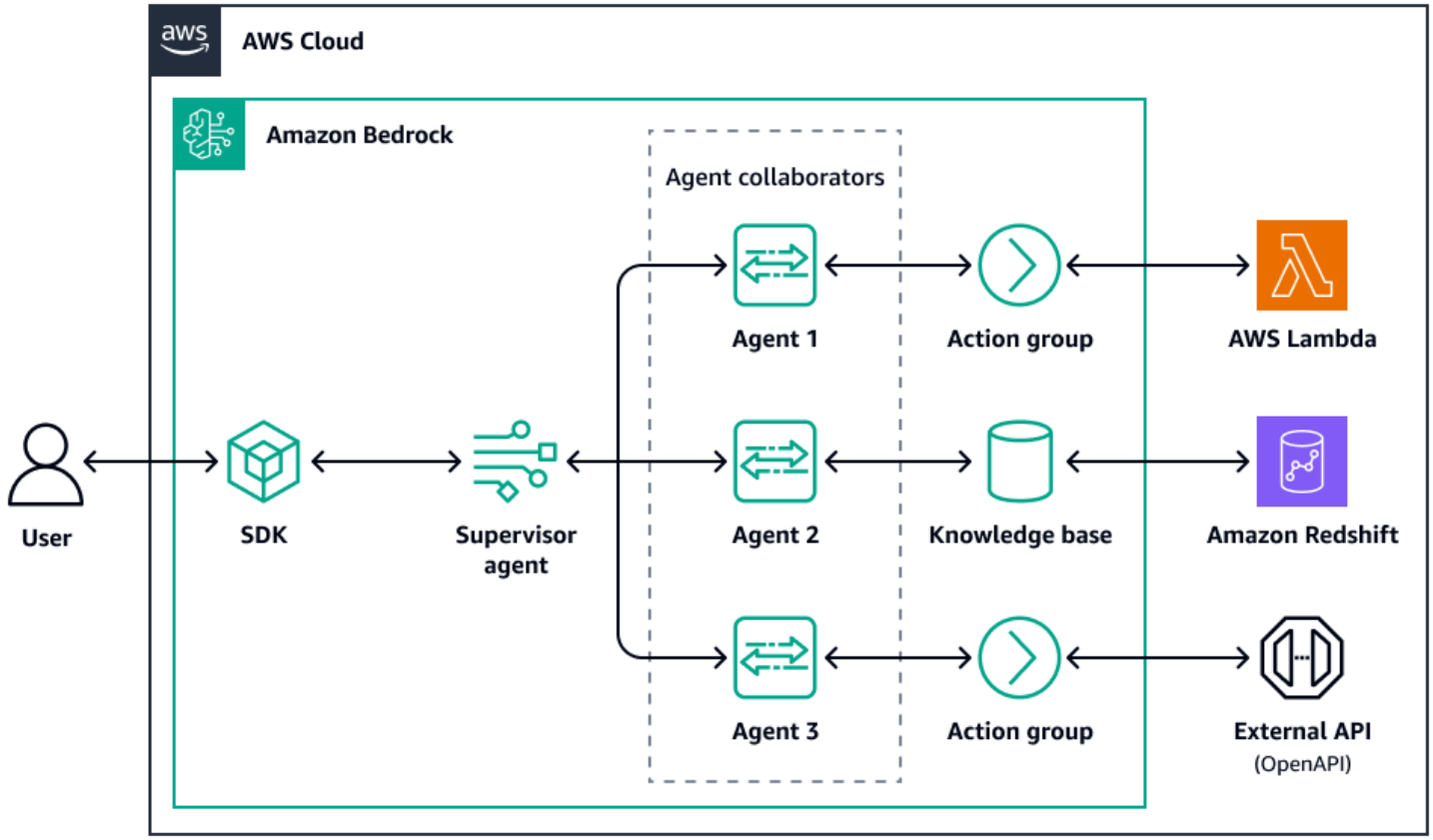
Supervisor

1. Ein Benutzer sendet die Abfrage „Erstellen Sie eine Projektbeschreibung und fassen Sie die fünf wichtigsten Wettbewerber zusammen“.
2. Der Orchestrator-Agent macht Folgendes:

- Weist einen Rechercheagenten mit der Suche nach Konkurrenzdaten zu
- Sendet die Rohergebnisse an einen Zusammenfassungsagenten
- Leitet die Ergebnisse an einen Mitarbeiter weiter, der einen Brief verfasst
- Kompiliert die endgültige Ausgabe für den Benutzer

Jeder Agent arbeitet unabhängig, aber der Orchestrator koordiniert die Aufgaben. Dies ist wie eine Lambda-Funktion, die Workflow-Aufgaben erledigt.

Das folgende Diagramm zeigt ein Beispiel für einen Supervisor:



1. Ein Benutzer reicht eine Aufgabe an einen Amazon Bedrock Supervisor-Mitarbeiter ein.
2. Der Supervisor-Agent unterteilt die Anfrage in Unteraufgaben für jeden Mitarbeitenden.
3. Jede Unteraufgabe wird einem Mitarbeiter-Agenten mit rollenspezifischen Eingabeaufforderungen oder Toolchains zugewiesen.
4. Worker Agents rufen externe APIs oder Tools über eine Aktionsgruppe auf.
5. Jeder Worker-Agent gibt die Ausgabe in einem strukturierten Format zurück.

6. Wenn alle Mitarbeiter ihre Ergebnisse zurückgeben, bewertet und fasst der Supervisor die endgültige Antwort zusammen und gibt sie zurück.

Diese Struktur ermöglicht Modularität, Anpassungsfähigkeit und Selbstbeobachtung bei komplexen, mehrstufigen Arbeitsabläufen für Agenten.

Imbissbuden

Während bei der Event-Orchestrierung (zum Beispiel AWS Step Functions) die Ausführung von Diensten durch zentrale Steuerung gesteuert wird, verwenden rollenbasierte Agentensysteme einen LLM-powered Orchestrator-Agenten, um das Ziel zu definieren, Unteraufgaben an Worker Agents zu delegieren und das Endergebnis zu synthetisieren.

In beiden Paradigmen macht der Orchestrator Folgendes:

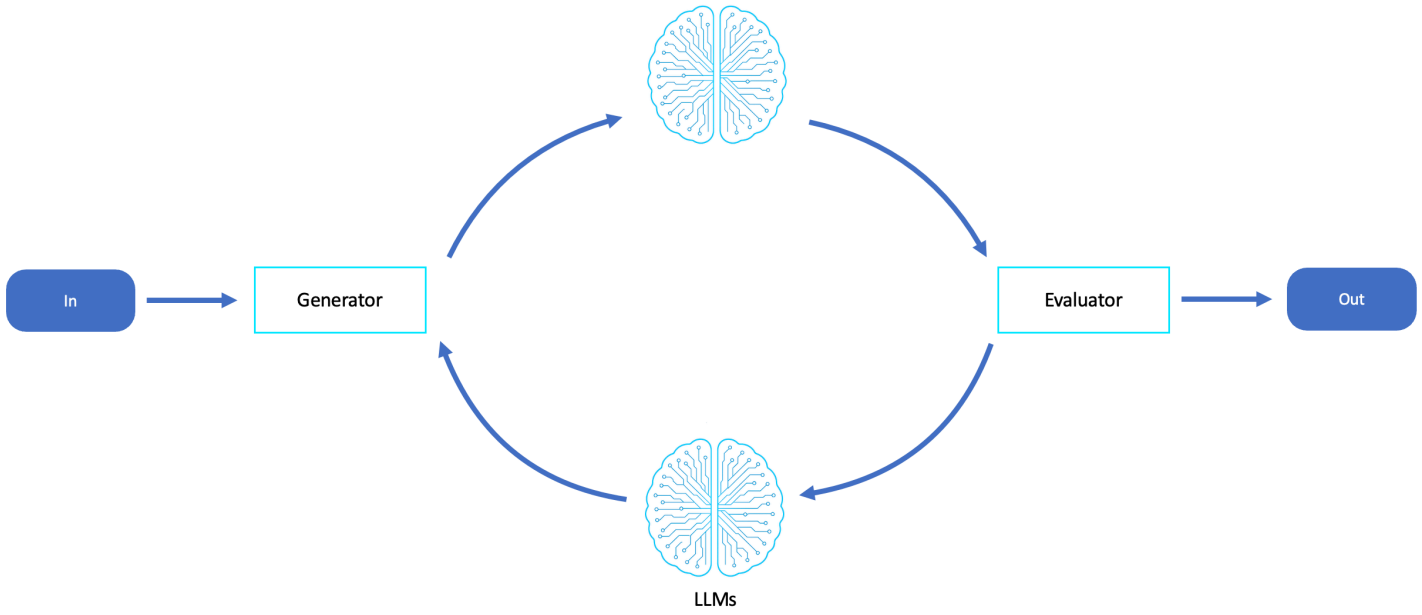
- Behält den Kontext und den Ausführungsablauf bei
- Verwaltet Verzweigungen, Sequenzierung und Fehlerbehandlung
- Erzeugt ein einheitliches Ergebnis aus verteilten Komponenten

Die behördliche Orchestrierung sorgt jedoch für mehr Argumentation, Anpassungsfähigkeit und semantische Delegierung. Dadurch eignet sie sich gut für offene, mehrdeutige und sich ständig weiterentwickelnde Aufgaben.

Schleifenmuster im Evaluator reflektieren/verfeinern

Aufgaben wie Codegenerierung, Zusammenfassung oder autonome Entscheidungsfindung profitieren in hohem Maße vom Feedback zur Laufzeit, sodass sich das System durch Beobachtung und Verfeinerung weiterentwickeln kann. Um dies zu operationalisieren, kann der Reflect-Refine-Zyklus als ereignisgesteuerter Feedback-Regelkreis implementiert werden — ein von der Systemtechnik inspiriertes Muster, das für autonome, intelligente Workflows angepasst ist.

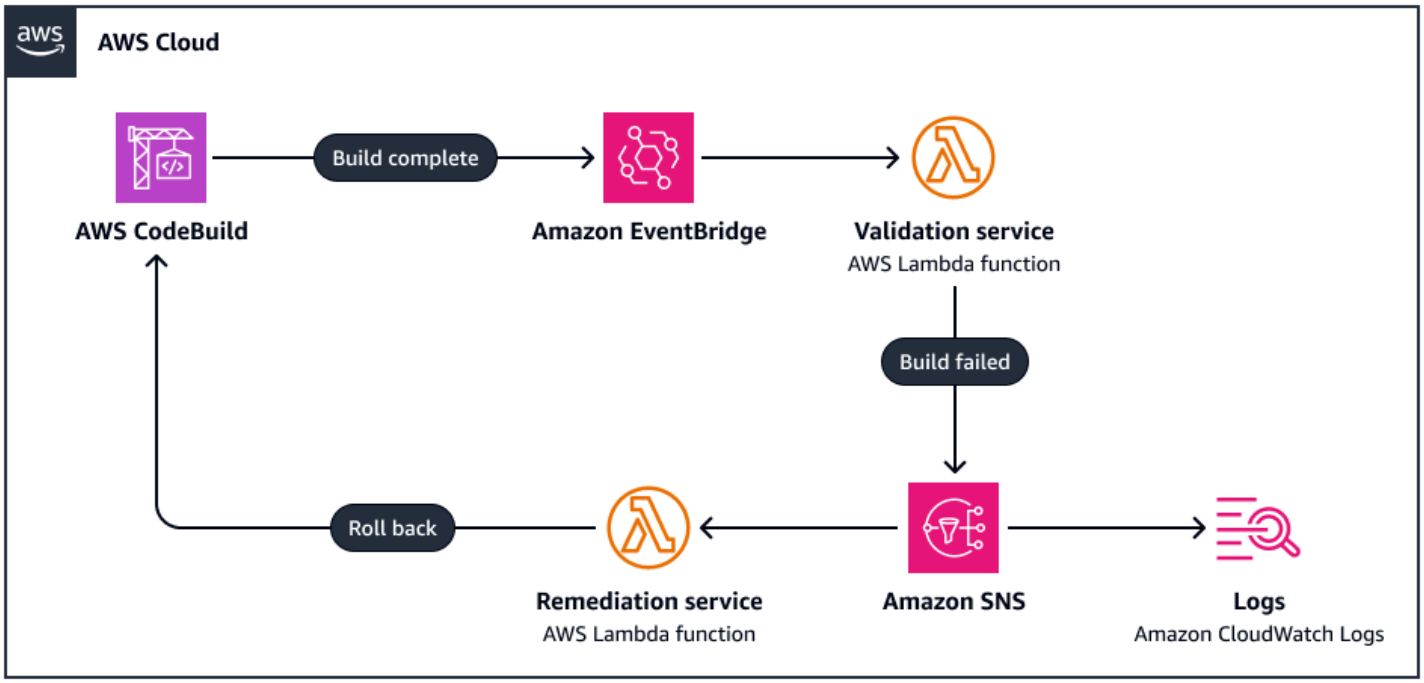
Das folgende Diagramm ist ein Beispiel für eine Reflect-Refine-Feedback-Schleife von Evaluatoren:



Regelkreis für Rückkopplung

Ein Feedback-Regelkreis ist ein Muster, das seine eigenen Leistungen und Verhaltensweisen überwacht, sie anhand definierter Kriterien oder eines gewünschten Zustands bewertet und dann seine Aktionen entsprechend anpasst. Diese Architektur ist von der Regelungstheorie inspiriert und ist grundlegend für Bereiche wie Automatisierung, kontinuierliche Integration und Continuous Delivery (CI/CD) -Pipelines sowie maschinelles Lernen.

Das folgende Diagramm ist ein Beispiel für einen Feedback-Regelkreis:



1. Eine Bereitstellungspipeline gibt ein BuildComplete-Ereignis aus.
2. Das Ereignis löst einen automatisierten Test- oder Evaluierungsjob aus, der den Build validiert.
3. Schlägt die Validierung fehl (z. B. aufgrund fehlgeschlagener Tests, Sicherheitsprobleme oder eines Richtlinienverstoßes), geht das System wie folgt vor:
   - Gibt ein BuildComplete-Ereignis aus
   - Protokolliert das Problem oder sendet eine Benachrichtigung
   - Löst eine Behebungs- oder Korrekturmaßnahme aus, z. B. ein Rollback, ein Patchen oder ein erneuter Versuch

Die Schleife wird fortgesetzt, bis ein akzeptables Ergebnis oder eine akzeptable Eskalation erzielt wird oder ein Timeout eintritt. Dieses Muster wird häufig für Folgendes verwendet:

- EventBridge Amazon-Regeln für die Weiterleitung von Ereignissen an Evaluierungs- oder Problembehebungsaufgaben
- AWS Step Functions für iterative Wiederholungslogik und Verzweigung der Evaluierungsergebnisse
- Amazon Simple Notification Service (Amazon SNS) oder CloudWatch Amazon-Alarme für Feedback-Auslöser und Benachrichtigungen
- AWS Lambda Funktionen oder Mitarbeiter in Containern, um Abhilfemaßnahmen zu ergreifen

Regelkreis für Rückkopplung (Evaluator)

Ein Evaluator-Workflow ist eine kognitive Feedback-Schleife, die von LLMs oder Argumentationsagenten unterstützt wird. Der Prozess besteht aus den folgenden Schritten:

1. Ein Generator-Agent oder LLM erzeugt eine Ausgabe (z. B. einen Plan, eine Antwort oder einen Entwurf).
2. Ein Gutachter überprüft das Ergebnis anhand einer Aufforderung zur Kritik oder einer Bewertungsrubrik.
3. Auf der Grundlage des Feedbacks überarbeitet der ursprüngliche Agent oder ein neuer Optimierer-Agent die Ausgabe.

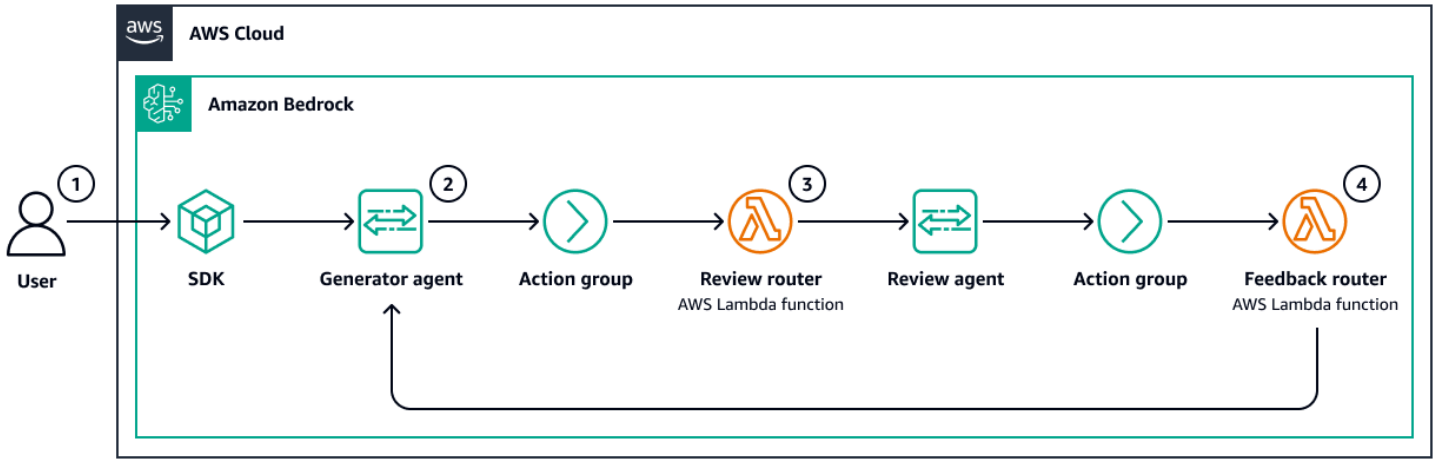
Die Schleife wird wiederholt, bis das Ergebnis eine Reihe von Kriterien erfüllt, genehmigt wurde oder ein Wiederholungslimit erreicht.

Evaluator

1. Ein Benutzer bittet einen Agenten, eine Richtlinienübersicht zu verfassen.
2. Der Generator-Agent entwirft sie.
3. Ein Gutachter überprüft den Umfang, den Ton und die rechtliche Richtigkeit.
4. Wenn die Antwort unzureichend ist, wird sie verfeinert und erneut eingereicht, bis die Feedback-Schleife zusammenläuft.

Dies ermöglicht eine Selbstbeurteilung, iterative Verfeinerung und adaptive Ausgabesteuerung — und das alles ohne menschliches Zutun.

Das folgende Diagramm ist ein Beispiel für einen Feedback-Regelkreis (Evaluator):



1. Ein Benutzer gibt eine Aufgabe aus (z. B. Entwurf einer Geschäftsstrategie).
2. Ein Amazon Bedrock-Agent generiert mithilfe eines LLM einen ersten Entwurf.
3. Ein zweiter Mitarbeiter (oder eine Folgeaufforderung) führt eine strukturierte Bewertung durch (z. B. „Bewerten Sie dieses Ergebnis nach Klarheit, Vollständigkeit und Tonalität“).
4. Wenn die Bewertung unter einen Schwellenwert fällt, wird die Antwort wie folgt überarbeitet:
   - Erneuter Aufruf des Generators mit eingebetteter Kritik
   - Senden des Feedbacks an einen spezialisierten Raffineriemitarbeiter
   - Iterieren, bis eine akzeptable Antwort erreicht ist

Optionale Komponenten wie AWS Lambda Controller AWS Step Functions können Feedback-Schwellenwerte, Wiederholungsversuche und Fallback-Strategien verwalten.

Imbissbuden

Während herkömmliche Feedback-Regelkreise Ereignisse, Metriken und Korrekturlogik verwenden, um das Systemverhalten zu validieren und anzupassen, verwenden behördliche Bewertungsschleifen Begründungsagenten, um Ergebnisse dynamisch zu bewerten, zu reflektieren und zu überarbeiten.

In beiden Paradigmen gilt:

- Die Ausgabe wird ausgewertet, nachdem sie generiert wurde
- Korrektur- oder Verbesserungsmaßnahmen werden auf der Grundlage von Feedback ausgelöst
- Das System passt sich kontinuierlich an eine Zielqualität oder ein Ziel an

Die agentische Version wandelt die statische Validierung in eine semantische Reflexion um und ermöglicht es Agenten, sich selbst zu verbessern, ihre eigene Effektivität zu bewerten.

Gestaltung agentischer Workflows auf AWS

Jedes Muster in diesem Handbuch kann mithilfe von erstellt werden. AWS-Services Amazon Bedrock-Agenten bieten Orchestrierung, Datenzugriff und Interaktionskanäle.

Komponente	AWS-Service	Zweck
LLM-Argumentation	Amazon Bedrock	Agentenlogik, Planung, Einsatz von Tools
Ausführung des Tools	AWS Lambda, Amazon ECS, Amazon SageMaker	Hosten Sie externe Tools für Agenten
Speicher und RAG	Amazon Bedrock Wissensdatenbank, Amazon S3, OpenSearch	Persistenter und semantischer Speicher
Orchestrierung	AWS Step Functions	Mehrstufige Aufgaben- und Agentenkoordination
Weiterleitung von Ereignissen	Amazon EventBridge, Amazon SQS	Entkoppeltes interagentenübergreifendes Messaging
Benutzeroberfläche	Amazon API Gateway AWS AppSync, SDK	Einstiegspunkte für Anwendungen oder Systeme
Überwachen	Amazon CloudWatch AWS X-Ray, AWS Distribution für OpenTelemetry	Beobachtbarkeit und Introspektion der Agenten

Schlussfolgerung

Agentische Workflow-Muster sind die nächste Evolutionsstufe ereignisgesteuerter Architekturen, bei denen die Geschäftslogik nicht statisch definiert, sondern mithilfe eines Large Language Model (LLM) — erweiterter Kognition — dynamisch begründet wird. Durch die Kombination traditioneller

cloudnativer Primitives mit LLM-Workflows und Agentendesignmustern können Unternehmen anpassungsfähige, intelligente und modulare Systeme aufbauen, die zielgerichtet reagieren und aus Erfahrungen lernen.

In diesen Mustern ist Amazon Bedrock das Tor zu agentischer Kognition und ermöglicht es LLM-based Agenten, auf Event-Workflows zuzugreifen, mit Tools und Speichern zu interagieren und strukturierte, nachvollziehbare und aufeinander abgestimmte Ergebnisse zu liefern.

Bei der Entwicklung und Bereitstellung von Agentensystemen bieten diese Workflow-Muster Blaupausen für den Aufbau autonomer, zusammensetzbarer KI-Architekturen. Diese Systeme basieren auf bewährten Methoden für serverlose Systeme und werden durch intelligente Basismodelle erweitert.

Dokumentverlauf

In der folgenden Tabelle werden wichtige Änderungen in diesem Leitfaden beschrieben. Um Benachrichtigungen über zukünftige Aktualisierungen zu erhalten, können Sie einen RSS-Feed abonnieren.

Änderung	Beschreibung	Datum
Erste Veröffentlichung	—	14. Juli 2025

Die vorliegende Übersetzung wurde maschinell erstellt. Im Falle eines Konflikts oder eines Widerspruchs zwischen dieser übersetzten Fassung und der englischen Fassung (einschließlich infolge von Verzögerungen bei der Übersetzung) ist die englische Fassung maßgeblich.