



でのエージェント AI のパターンとワークフロー AWS

AWS 規範ガイド



AWS 規範ガイド: でのエージェント AI のパターンとワークフロー

AWS

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

Amazon の商標およびトレードドレスは Amazon 以外の製品およびサービスに使用することはできません。また、お客様に誤解を与える可能性がある形式で、または Amazon の信用を損なう形式で使用することもできません。Amazon が所有していないその他のすべての商標は Amazon との提携、関連、支援関係の有無にかかわらず、それら該当する所有者の資産です。

Table of Contents

序章	1
対象者	1
目的	1
このコンテンツシリーズについて	2
エージェントパターン	3
基本的な推論エージェント	4
アーキテクチャ	4
説明	5
機能	6
制限事項	6
一般的なユースケース	6
実装のガイド	7
概要	7
アーキテクチャ	7
説明	8
機能	9
一般的なユースケース	9
実装のガイド	9
概要	10
関数を呼び出すためのツールベースのエージェント	10
アーキテクチャ	10
説明	11
機能	12
一般的なユースケース	12
実装のガイド	12
概要	13
サーバー用のツールベースのエージェント	13
アーキテクチャ	13
説明	14
機能	15
一般的なユースケース	15
実装のガイド	15
概要	16
コンピュータ使用エージェント	16

アーキテクチャ	16
説明	17
機能	18
一般的なユースケース	18
実装のガイド	19
概要	19
コーディングエージェント	19
アーキテクチャ	19
説明	20
機能	21
一般的なユースケース	21
実装のガイド	22
概要	22
音声エージェント	22
アーキテクチャ	22
説明	23
機能	24
一般的なユースケース	24
実装のガイド	25
概要	25
ワークフローオーケストレーションエージェント	25
アーキテクチャ	26
説明	26
機能	27
一般的なユースケース	27
実装のガイド	27
概要	28
メモリ拡張エージェント	28
アーキテクチャ	28
説明	29
機能	30
一般的なユースケース	30
メモリ拡張エージェントの実装	30
メモリ注入プロンプトの実装	31
概要	31
シミュレーションおよびテスト床エージェント	32

アーキテクチャ	32
説明	33
機能	34
一般的なユースケース	34
実装のガイド	35
概要	36
オブザーバーエージェントとモニタリングエージェント	36
アーキテクチャ	37
説明	37
機能	38
一般的なユースケース	38
実装のガイド	38
概要	39
マルチエージェントコラボレーション	39
説明	41
機能	42
一般的なユースケース	42
実装のガイド	42
概要	43
結論	43
重要事項	44
LLM ワークフロー	45
LLM で拡張された認識の概要	45
プロンプト連鎖のワークフロー	46
説明	47
機能	47
一般的なユースケース	48
ルーティングのワークフロー	48
機能	49
一般的なユースケース	49
並列化のワークフロー	49
機能	51
一般的なユースケース	51
オーケストレーションのワークフロー	51
機能	52
一般的なユースケース	52

評価者とリフレクション/絞り込みループのワークフロー	53
一般的なユースケース	54
機能	54
結論	54
エージェントワークフローパターン	56
イベント駆動型システムから認識拡張型システムへ	56
イベント駆動型アーキテクチャ	56
認知拡張ワークフロー	57
コアインサイト	59
Saga パターンのプロンプト連鎖	59
Saga コレオグラフィ	60
プロンプトの連鎖パターン	60
エージェントコレオグラフィ	61
重要事項	63
動的ディスパッチパターンのルーティング	63
動的ディスパッチ	64
LLM ベースのルーティング	65
エージェントルーター	66
重要事項	67
並列化パターンと散布図パターン	67
散布-収集	68
LLM ベースの並列化 (散布図認識)	70
エージェント並列化	70
重要事項	71
Saga オーケストレーションパターン	71
イベントオーケストレーション	72
ロールベースのエージェントシステム (オーケストレーター)	73
スーパーバイザー	73
重要事項	75
評価者の反射/絞り込みループパターン	75
フィードバックコントロールループ	76
フィードバックコントロールループ (評価者)	77
評価者	77
重要事項	78
でのエージェントワークフローの設計 AWS	79
結論	80

ドキュメント履歴	81
.....	lxxxii

でのエージェント AI のパターンとワークフロー AWS

Aaron Sempf と Andrew Hooker、Amazon Web Services

2025 年 7 月 ([ドキュメント履歴](#))

組織は、エージェントパターンと呼ばれる新しいアーキテクチャ分野を使用して、動的でマルチドメインの問題を解決するために、大規模言語モデル (LLMs) とソフトウェアエージェントを採用しています。エージェントパターンは、さまざまなコンテキストで目標指向の AI エージェントを設計およびオーケストレーションするために使用される基本的な設計図とモジュール構造です。

対象者

このガイドは、静的ロジック、シンボリックロジック、決定論的オートメーションを超えるインテリジェントなアプリケーションを構築したいと考えているアーキテクト、デベロッパー、製品リーダーを対象としています。

目的

このガイドでは、制御可能で目標に沿った状態で自律的に動作する AI エージェントシステムの設計フレームワークと実装アプローチを提供します。イベント駆動型のアーキテクチャパターンをさまざまなエージェント代替案と接続し、クラウドネイティブアーキテクチャを使用して本稼働グレードのエージェントシステムを構築する方法を示します。このガイドでは、以下のトピックについて説明します。

- エージェントパターン – エージェントパターンは、個々のエージェントの構造と動作を記述する再利用可能な設計テンプレートです。これには、推論エージェント、検索拡張エージェント、コーディングエージェント、音声インターフェイス、ワークフローオーケストレーター、共同マルチエージェントシステムが含まれます。各パターンは、エージェントが認識、理由、行動、学習し、マッピングする方法を示しています AWS のサービス。
- LLM ワークフロー – ワークフローは、エージェントが推論のために LLMs を使用方法に焦点を当てています。プロンプト戦略と計画メカニズムを検討し、LLMs を使用してテキストを生成するだけでなく、エージェントループ内で構造化され、解釈可能で、信頼性の高い動作を促進する方法について概説します。
- エージェントワークフローパターン – ワークフローパターンは、複数のエージェント、ツール、環境がどのように相互作用して自律システムを形成するかを記述します。これには、タスクオーケ

ストレーション、サブエージェントの委任、イベントベースの調整、オブザーバビリティ、コントロールのパターンが含まれます。これらの側面は、スケーラブル、構成可能、監査可能な AI アーキテクチャを促進します。

このコンテンツシリーズについて

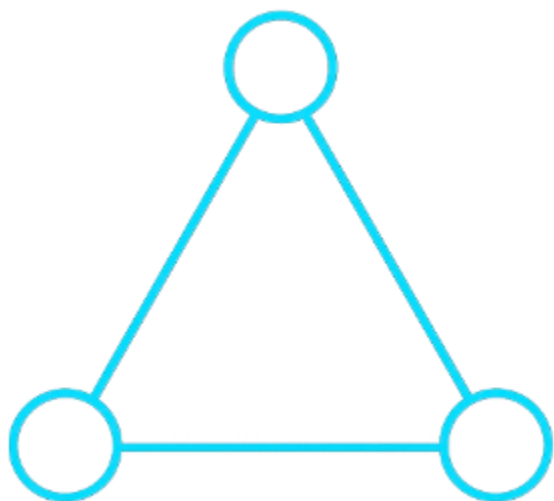
このガイドは、でのエージェント AI に関するシリーズの一部です AWS。詳細およびこのシリーズの他のガイドについては、AWS 「規範ガイダンス」ウェブサイトの [「エージェント AI」](#) を参照してください。

エージェントパターン

エージェントパターンは、特定のドメイン、ユースケース、複雑さのレベルに合わせて調整できる、再利用可能な構成可能な構成要素です。ただし、エージェントシステムは従来のアプリケーションとは異なります。すべての AI エージェント設計の中心となるのは、次の 3 つの基本原則に基づく概念モデルです。

- 非同期 – エージェントは疎結合のイベントの多い環境で動作します。
- 自律性 – エージェントは人間や外部の制御なしで独立して動作します
- エージェンシー – エージェントは、特定の目標に向けて、ユーザーまたはシステムに代わって目的を持って行動します。

次の図の三角形は、ソフトウェアエージェントの中核的な構成要素である認識、理由、アクションを表しています。これにより、エージェントシステムは環境内で監視、意思決定、行動できるようになります。



設計上、エージェントパターンは AI システムを構築するためのモジュール設計言語を提供します。つまり、アクセス可能、運用可能、拡張可能、本番稼働準備ができています。これらのシステムを設計するには、次の 3 つの相互に関連するディメンションに細心の注意が必要です。詳細については、このガイドの後半で説明します。

このセクションの内容

- [基本的な推論エージェント](#)
- [関数を呼び出すためのツールベースのエージェント](#)

- [サーバー用のツールベースのエージェント](#)
- [コンピュータ使用エージェント](#)
- [コーディングエージェント](#)
- [音声エージェント](#)
- [ワークフローオーケストレーションエージェント](#)
- [メモリ拡張エージェント](#)
- [シミュレーションおよびテスト床エージェント](#)
- [オブザーバーエージェントとモニタリングエージェント](#)
- [マルチエージェントコラボレーション](#)

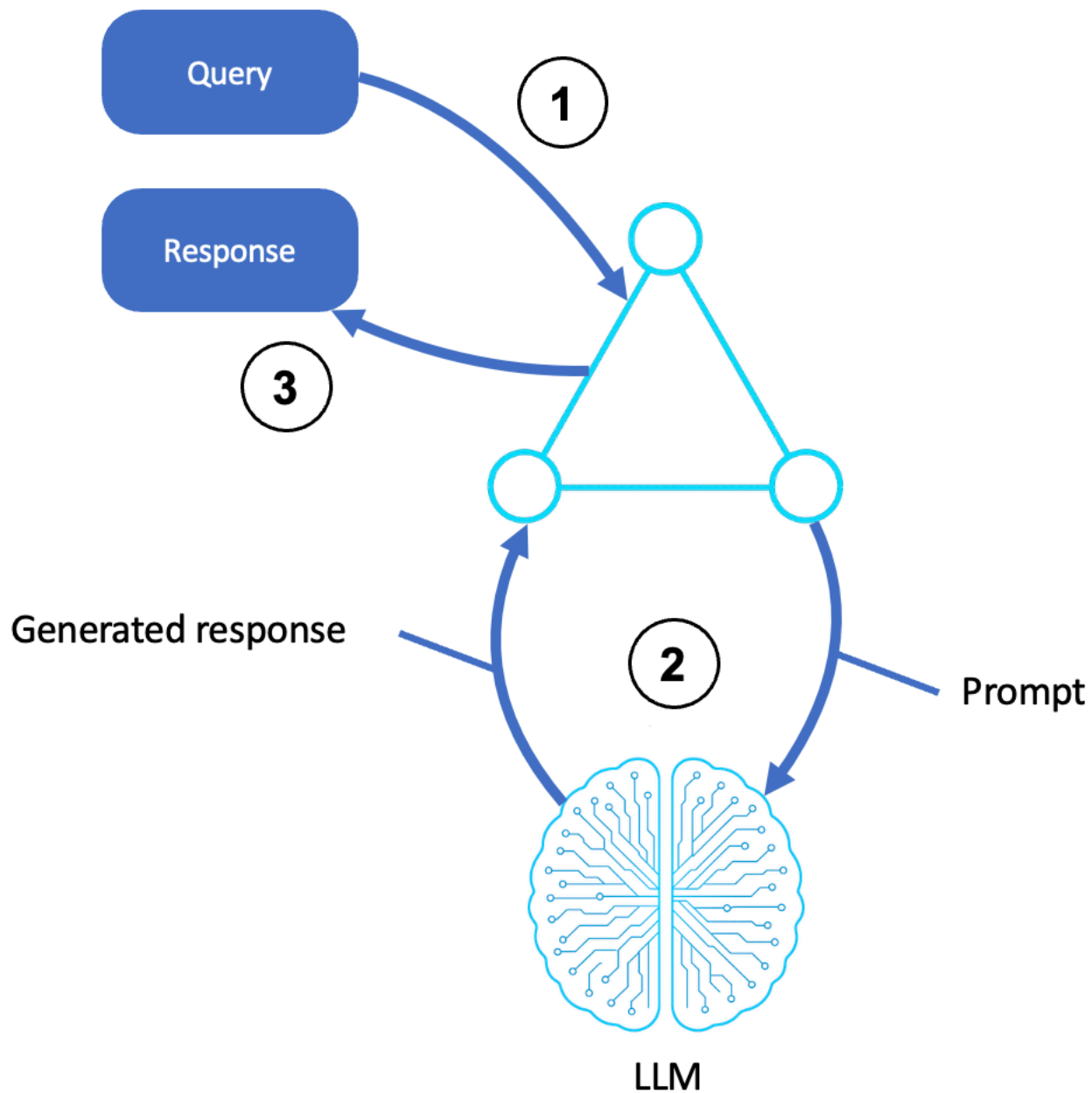
基本的な推論エージェント

基本的な推論エージェントは、クエリに応答して論理推論または意思決定を実行するエージェント AI の最も簡単な形式です。ユーザーまたはシステムからの入力を受け入れ、クエリを処理し、構造化プロンプトを使用してレスポンスを生成します。

このパターンは、特定のコンテキストに基づいて単一ステップの推論、分類、または要約を必要とするタスクに役立ちます。メモリ、ツール、または状態管理を使用しないため、ステートレスで軽量で、大規模なワークフローで高度に構成できます。

アーキテクチャ

基本的な推論エージェントのフローを次の図に示します。



説明

1. 入力を受け取る

- ユーザー、システム、またはアップストリームエージェントがクエリまたは命令を送信します。
- 入力はエージェントシェルまたはオーケストレーションレイヤーに引き渡されます。
- このステップには、前処理、プロンプトテンプレート作成、目標の特定が含まれます。

2. LLM を呼び出します

- エージェントはクエリを構造化されたプロンプトに変換し、LLM に送信します (Amazon Bedrock 経由など)。
- LLM は、事前トレーニング済みの知識とコンテキストを使用して、プロンプトに基づいてレスポンスを生成します。
- 生成された出力には、推論ステップ (chain-of-thought)、最終回答、またはランク付けされたオプションが含まれる場合があります。

3. レスポンスを返します。

- 生成された出力は、エージェントのインターフェイスに中継されます。
- これには、フォーマット、後処理、または API レスポンスが含まれます。

機能

- 自然言語または構造化された入力をサポート
- プロンプトエンジニアリングを使用して動作をガイドします
- ステートレスでスケーラブル
- UI、CLI、APIs、パイプラインに埋め込むことができます

制限事項

- メモリや履歴認識がない
- 外部ツールやデータソースとやり取りしない
- 推論時に LLM が知っているものに限定

一般的なユースケース

- 会話の質問と回答
- ポリシーの説明と概要
- 意思決定のためのガイダンス
- 軽量で自動化されたチャットボットフロー
- 分類、ラベル付け、スコアリング

実装のガイド

次のツールとサービスを使用して、基本的な推論エージェントを作成できます。

- Amazon Bedrock for LLM 呼び出し (Anthropic、AI21、Meta)
- Amazon API Gateway または AWS Lambda を使用してステートレスマイクロサービスとして公開する
- パラメータストアに保存されているプロンプトテンプレート AWS Secrets Manager、またはコードとして保存されているプロンプトテンプレート

概要

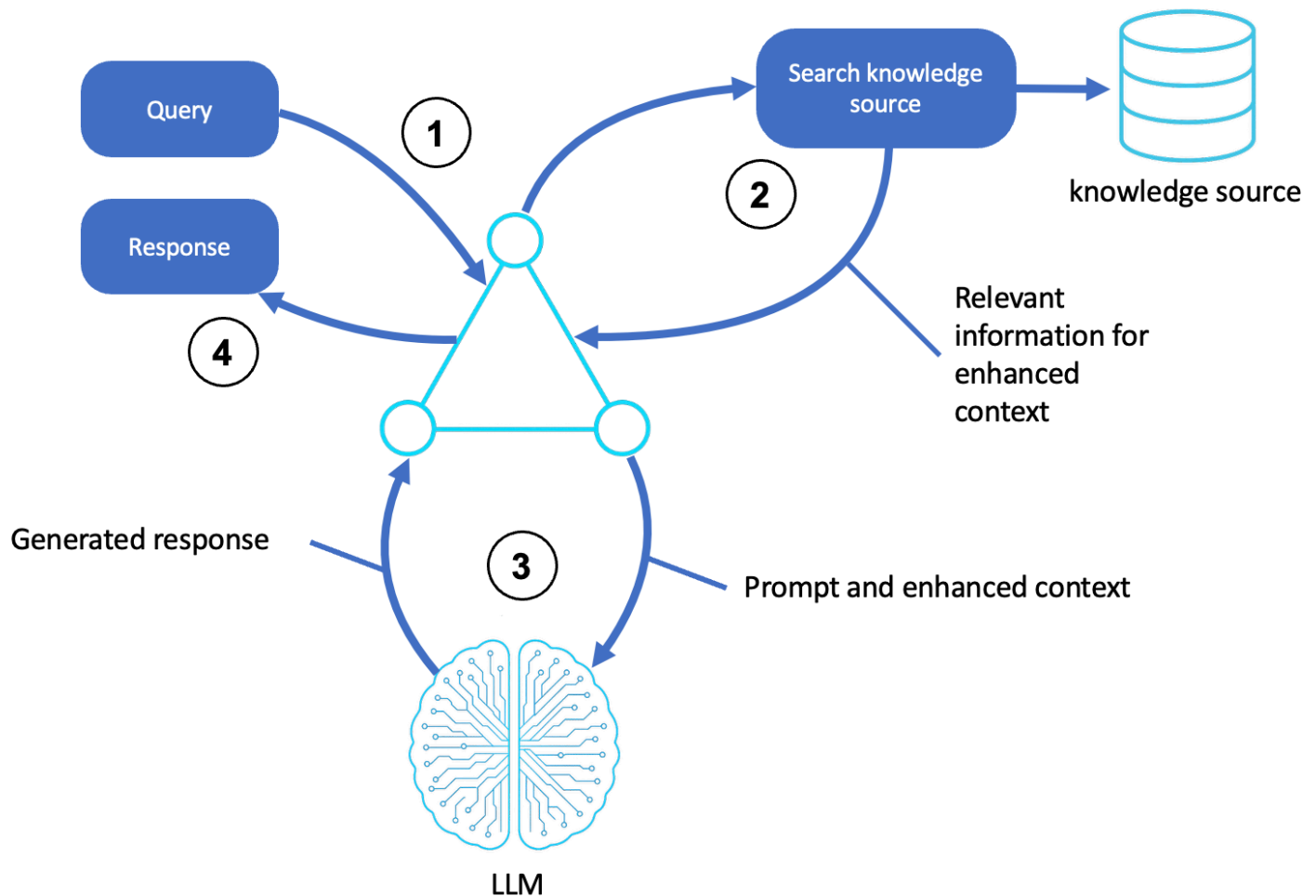
基本的な推論エージェントは、シンプルな構造のため、基本的なものです。目標をインテリジェント出力につながる推論パスに変換するコア機能を備えています。このパターンは、多くの場合、ツールベースのエージェントや取得拡張生成 (RAG) を使用するエージェントなど、高度なパターンの開始点です。また、信頼性の高い大規模なワークフローのモジュールコンポーネントでもあります。

エージェント RAG

Retrieval-augmented generation (RAG) は、情報の取得とテキスト生成を組み合わせ、正確でコンテキストに応じたレスポンスを作成する手法です。RAG を使用すると、エージェントは LLM をエンゲージする前に関連する外部情報を取得できます。エージェントの意思決定を up-to-date、事実、またはドメイン固有の情報に基づいて判断することで、エージェントの有効なメモリと推論の精度を拡張します。事前トレーニング済みの重みのみに依存するステートレス LLMs とは対照的に、RAG にはコンテキストを使用してプロンプトを動的に強化する外部ナレッジ検索レイヤーがあります。

アーキテクチャ

RAG パターンのロジックを次の図に示します。



説明

1. クエリを受信する

- ユーザーまたはアップストリームシステムは、クエリまたは目標をエージェントに送信します。
- エージェントシェルはリクエストを受け入れ、推論のプロンプトとしてフォーマットします。

2. 外部ソースを検索します。

- エージェントは、クエリから概念とインテントを識別します。
- セマンティック検索またはキーワードマッチングを使用して、ベクトルストア、データベース、ドキュメントインデックスなどのナレッジソースをクエリします。
- 最も関連性の高いパッセージ、ドキュメント、またはエンティティは、次のステップで使用するために取得されます。

3. コンテキストレスポンスを生成します。

- エージェントは、取得した情報でプロンプトを強化し、LLM のコンテキスト拡張入力を形成します。
- LLM は、生成推論 (chain-of-thought やリフレクションなど) を使用して入力を処理し、正確なレスポンスを生成します。

4. 最終出力を返します。

- エージェントは、出力を任意の通信ヘッダーまたは必要な形式にラップして準備し、ユーザーまたは呼び出しシステムに返します。
- (オプション) 取得したドキュメントと LLM 出力は、今後のクエリのためにログに記録、スコアリング、メモリに保存される場合があります。

機能

- ロングテールドメインやエンタープライズ固有のドメインでも事実に基づく出力
- モデルを微調整しないメモリ拡張
- 各クエリとユーザーの状態に基づく動的コンテキスト
- ベクトルデータベース、セマンティックインデックス、メタデータフィルタリングとの完全な互換性

一般的なユースケース

- エンタープライズナレッジアシスタント
- 規制コンプライアンスボット
- カスタマーサポートの副操縦士
- 検索が強化されたチャットボット
- デベロッパードキュメントエージェント

実装のガイド

次のツールとサービスを使用して、RAG を使用するエージェントを作成します。

- Amazon Bedrock for LLM 呼び出し
- ドキュメントまたは構造化データ検索用の Amazon Kendra、OpenSearch、または Amazon Aurora

- ドキュメントストレージ用の Amazon Simple Storage Service (Amazon S3)
- AWS Lambda 検索、プロンプト、LLM 推論をオーケストレーションする
- エージェントとのナレッジベースの統合 (メモリプラグイン、セマンティックリトリバー、または Amazon Bedrock を使用)

概要

エージェント RAG は、静的モデル推論を動的な実世界のインテリジェンスに接続します。これにより、エージェントは知らないものを検索し、取得した知識から回答を合成し、信頼性の高い監査可能な応答を生成できます。

RAG パターンは、再トレーニングなしでナレッジアクセスをスケールするインテリジェントなエージェントを構築するための基盤です。多くの場合、ツールの使用、計画、長期メモリを含む、より複雑なオーケストレーションパターンの先駆けとなります。

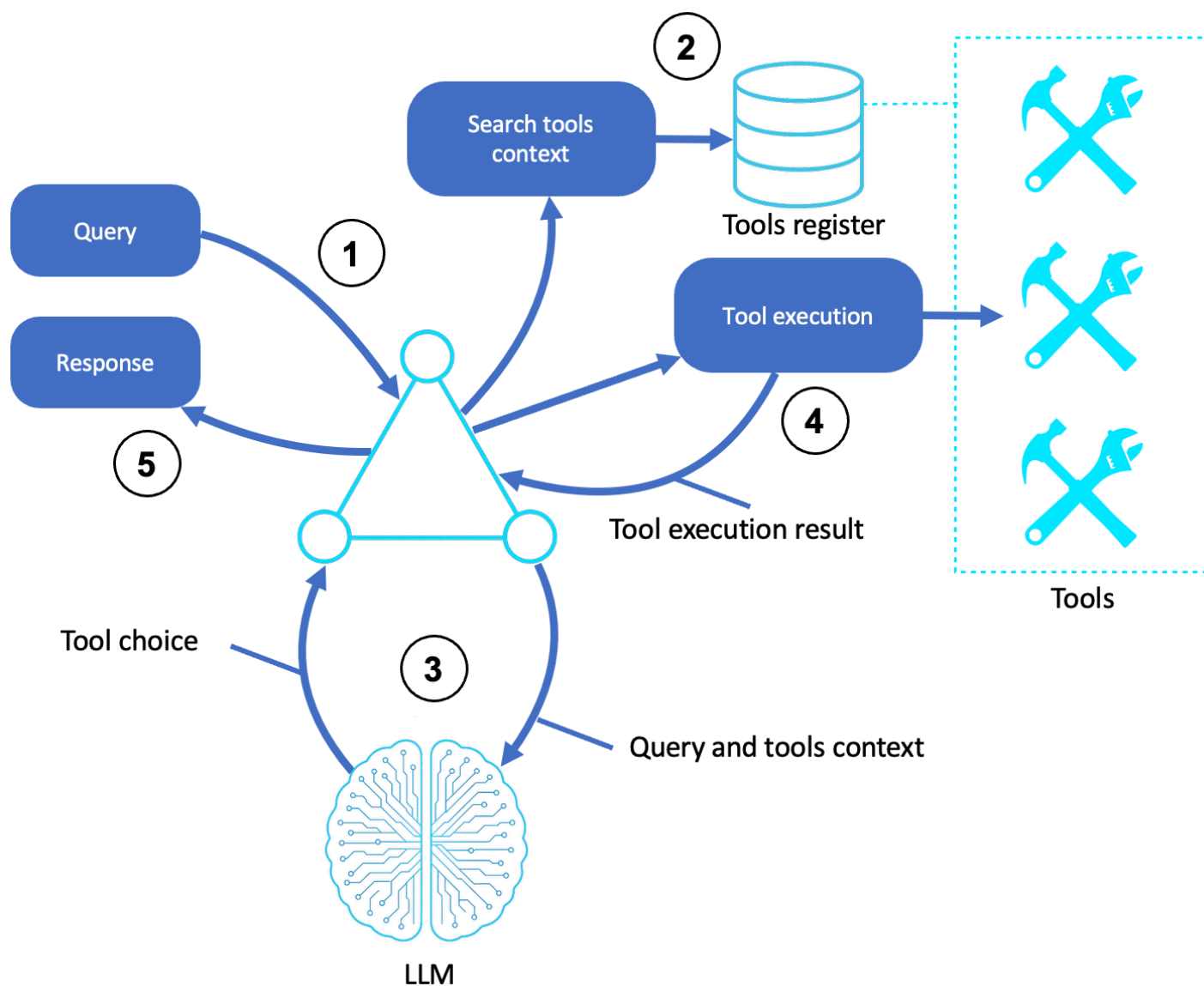
関数を呼び出すためのツールベースのエージェント

ツールベースのエージェントは、外部関数または APIs で、言語のみの推論を超えるタスクを完了することで、推論エージェントの機能を拡張します。このパターンでは、LLM を使用して使用するツールを決定し、呼び出し引数を生成し、ツールの出力を推論ループに組み込みます。

このパターンにより、エージェントは単にレスポンスを提供するのではなく、行動できるようになります。ツールインターフェイスは、算術計算やデータベース検索から外部 APIs やクラウドサービスまで、呼び出し可能な機能を表します。

アーキテクチャ

関数を呼び出すためのツールベースのエージェントを次の図に示します。



説明

1. クエリを受信する

- エージェントは、ユーザーまたは呼び出しシステムから自然言語クエリまたはタスクを受け取ります。

2. ツールの検索

- エージェントは内部メタデータまたはツールレジストリを使用して、使用可能なツール、スキーマ、関連機能を検索します。

3. ツールを選択して呼び出す

- LLM は、クエリとツールのメタデータ (関数名、入力タイプ、説明など) をプロンプトで受け取ります。
- 最も関連性の高いツールを選択し、入力引数を構築し、構造化された関数呼び出しを返します。

4. 選択したツールを実行します

- エージェントシェルまたはツールランナーは、選択した関数を実行し、結果 (API 出力、データベース値、計算など) を返します。

5. レスポンスを返します。

- LLM は、結果をエージェントに直接渡すか、更新されたプロンプトの一部として渡します。次に、自然言語の結果を返します。

機能

- タスクコンテキストに基づく動的ツールの選択
- スキーマベースのプロンプト (OpenAPI、JSON スキーマ、AWS 関数インターフェイス)
- 結果の解釈と出力の推論への連鎖
- ステートレスまたはセッション対応オペレーション

一般的なユースケース

- 外部データアクセスを持つ仮想アシスタント
- 財務計算ツールと見積りツール
- API ベースのナレッジワーカー
- を呼び出す LLMs AWS Lambda、Amazon SageMaker エンドポイント、SaaS サービス

実装のガイド

関数を呼び出すためのツールベースのエージェントを作成するには、以下を使用します。

- 関数呼び出しをサポートする Amazon Bedrock (Anthropic Claude)
- AWS Lambda ツール実行バックエンドとしての
- Amazon API Gateway またはツールオーケストレーション AWS Step Functions 用

- コンテキスト対応ツールメタデータ用の Amazon DynamoDB または Amazon Relational Database Service (Amazon RDS)
- 状態をマッピングして出力をルーティング AWS Step Functions する Amazon EventBridge パイプラインまたは

概要

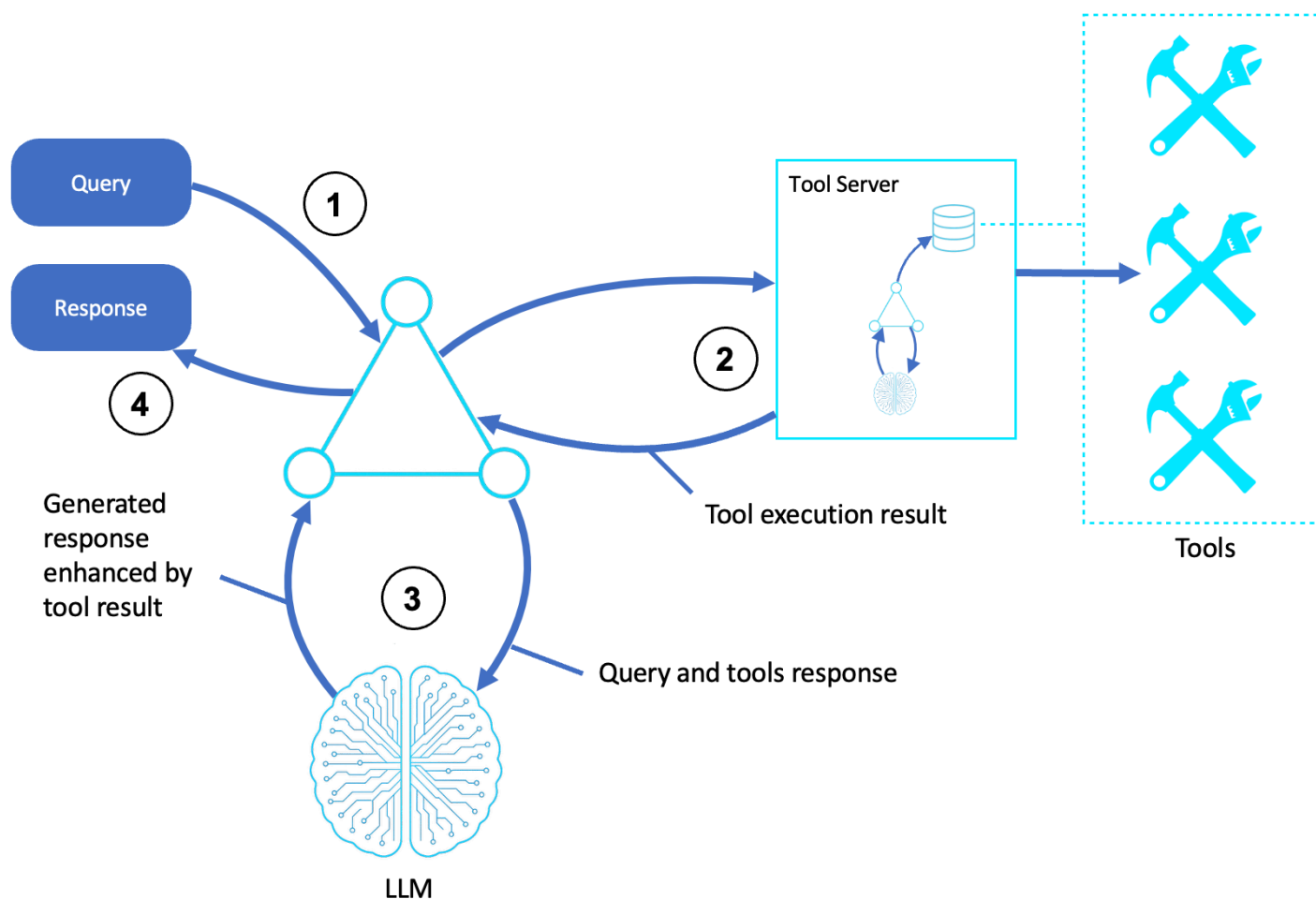
ツールベースの関数呼び出しエージェントは、言語の理解からアクションの実行への移行を表します。これらのエージェントは、LLM 推論を維持し、パッシブアシスタントをタスクを完了し、サービスにアクセスし、ビジネスオペレーションを統合するシステムに変換しながら、動的なコンテキスト対応ツールを呼び出します。このパターンは、特に宣言型スキーマ、認可フレームワーク、マルチエージェントシステムと組み合わせる場合、エンタープライズ設定におけるエージェント AI の重要なコンポーネントです。

サーバー用のツールベースのエージェント

サーバー用のツールベースのエージェントは、ツールの実行を、ツール、スクリプト、複合エージェント専用のランタイム環境を持つ外部サーバーに委任することで、関数呼び出しエージェントを強化します。エージェントグループが選択および呼び出すインライン関数呼び出しとは異なり、サーバーベースのエージェントはロジックと実行パイプラインを他のエージェントまたはシステムにアウトソースします。これにより、マルチツールの連鎖、分離された実行、特殊な推論などの高度な機能が提供されます。ツールサーバーは、ツール自体に個別の AI モデル、ビジネスルール、または環境が含まれる可能性がある、複雑でステートフル、またはリソース集約型のアクションに最適です。

アーキテクチャ

サーバー用のツールベースのエージェントのパターンを次に示します。



説明

1. クエリを受信する

- ユーザーまたはシステムは、エージェントシェルにリクエストを送信します。
- エージェントはクエリを解釈し、ツールサーバーにディスパッチする準備をします。

2. ツールサーバープロセスを実行します。

- エージェントは、構造化パラメータとともにタスクをツールサーバーに送信します。
- その後、ツールサーバーは以下を行うことができます。
 - 専用コンピューティングシステム (コンテナ AWS Lambda、Amazon SageMaker など) でスクリプトまたはロジックを実行する
 - LLM 推論で独自のサブエージェントを使用してツールを選択して実行する
 - 依存関係、再試行、またはマルチステップ実行フローを管理する
 - タスクの完了時に結果をプライマリエージェントに出力する

3. ツール出力で LLM 推論を使用する

- エージェントは LLM を呼び出し、元のクエリとツールサーバーの結果をプロンプトの一部として渡します。
- LLM は、新しく取得した情報を組み込んだレスポンスを合成します。

4. レスポンスを返します。

- エージェントは、自然言語または構造化されたレスポンスをユーザーまたは呼び出しシステムに返します。
- (オプション) 結果はメモリまたは監査ログに保存できます。

機能

- ツールはプライマリエージェント実行ループの外部で呼び出されます
- ツールの実行には、LLM 呼び出し、ロジックチェーン、またはサブエージェントが含まれる場合があります
- エージェントはツールラッパーだけでなく、コントローラーまたはディスパッチャーとして機能します
- ロジックのコンポジビリティ、スケーラビリティ、分離を有効にする

一般的なユースケース

- モデルチェーンのオーケストレーション (LLM、ビジョン、コードの組み合わせなど)
- AI 駆動型オートメーションパイプライン
- スクリプトランナーを使用する DevOps アシスタントエージェント
- 複雑な財務計算、シミュレーション、または最適化エージェント
- マルチモーダルツール (オーディオ、ドキュメント、アクションを組み合わせるなど)

実装のガイド

このパターンは、以下を使用して構築できます AWS のサービス。

- Amazon Bedrock (エージェントホストと LLM 推論)
- AWS Lambda ツールサーバーのランタイムとしての、Amazon ECS AWS Fargate、または Amazon SageMaker エンドポイント

- Amazon API Gateway または AWS App Runner を使用してツールサーバー APIs
- デカップリングされたagent-to-toolへのメッセージング用の Amazon EventBridge
- AWS Step Functions ツールサーバーでマルチエージェントロジックを作成 AWS AppFabric するための または

概要

サーバーを使用するツールベースのエージェントは、高度にモジュール化され、スケーラブルです。決定ロジックを実行から切り離します。これにより、複雑なアクションや機密性の高いアクションを他のシステムにオフロードしながら、プライマリエージェントを軽量に保つことができます。これは、エンタープライズグレードのエージェント AI、特にガバナンス、オブザーバビリティ、分離、動的構成、またはそれらの任意の組み合わせを必要とする環境では重要です。

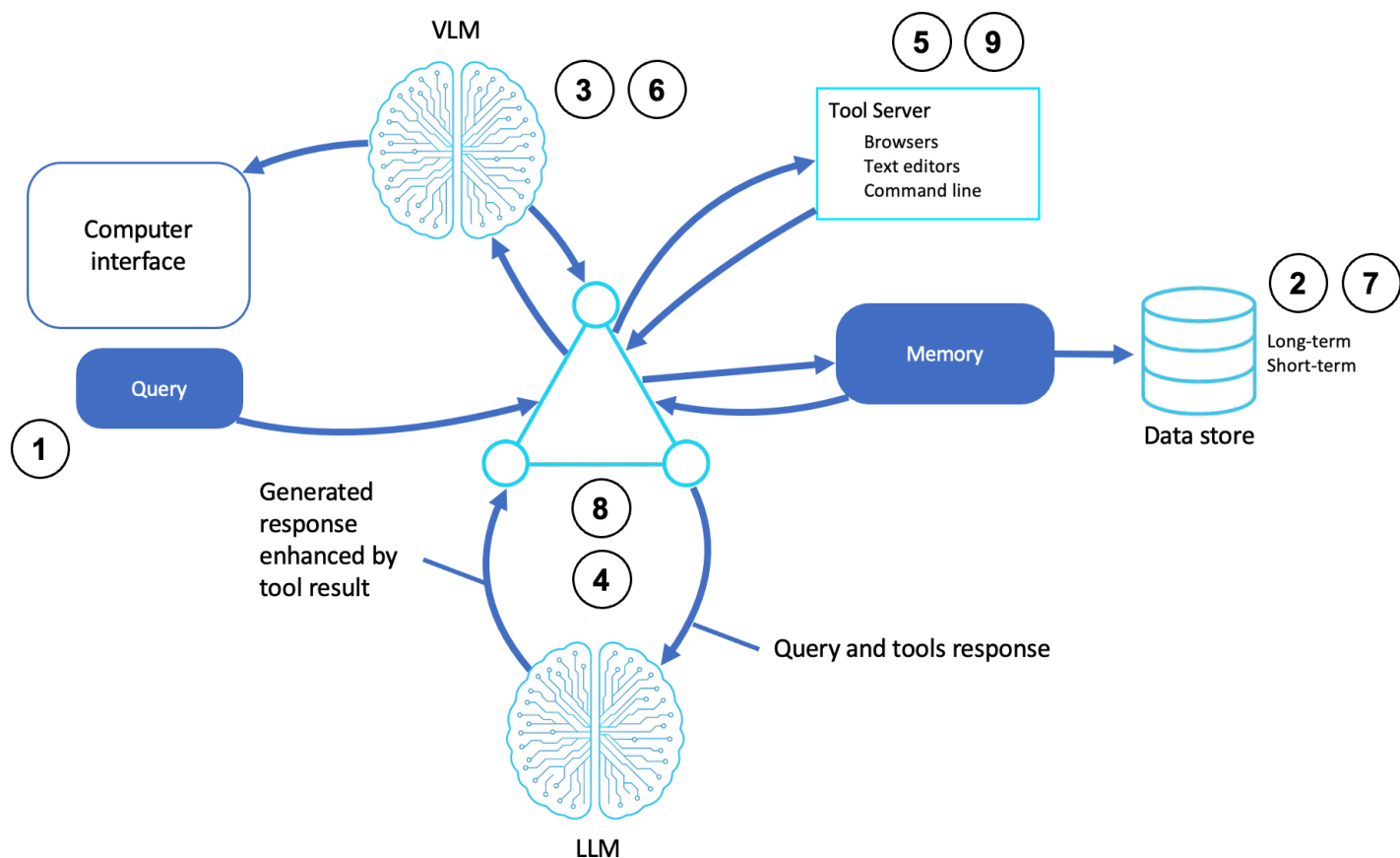
コンピュータ使用エージェント

コンピュータを使用するエージェントは、ブラウザ、ターミナル、ファイルシステム、アプリケーションなどのデジタル環境をシミュレートまたは制御できます。これらのエージェントは、LLM 推論、ビジュアル言語モデル (VLMs)、およびコマンドを実行したり入力イベントをシミュレートしたりするツールサーバーを組み合わせて、ユーザーのインテントを解釈し、ビジュアルインターフェイスとテキストインターフェイスを操作し、目標指向のアクションを実行します。

このパターンは、エージェントがアシスタントとしてだけでなく、人間と同じようにアクションを実行するプロキシとしても機能する実用的な AI オートメーションにとって重要です。多くの場合、同じツールと環境を使用します。

アーキテクチャ

コンピュータ使用のエージェントパターンを次の図に示します。



説明

1. クエリを受信する

- タスクまたはリクエストは、UI、API、または自然言語インターフェイスを介して提供されます。

2. メモリにアクセスします

- エージェントは短期および長期のメモリを取得して、過去のコマンド、目標、システム状態を再現します。

3. ビジュアルコンテキストを分析する

- VLM は、コンピュータ画面、システム状態、または UI 要素を観察して、特定のコンテキストを理解し、実用的な項目を特定します。

4. LLM 経由の理由

- LLM は、クエリ、メモリ状態、ツール、サーバーレスポンスを組み合わせ、次のアクションを決定します。

5. ツールサーバーを操作する

- エージェントは、サーバーでホストされているツールを呼び出します。これには、以下が含まれる場合があります。
- ブラウザ (ヘッドレス Chrome など) とシェル環境
- テキストエディタとコードエディタ
- カスタムスクリプトインターフェイス

6. ビジュアル入力を更新します。

- システム UI が変更されたり、さらに監視が必要な場合、VLM は画面の状態またはテキストバッファを再分析することがあります。

7. メモリの更新

- 新しいインサイト、システム状態、またはユーザーフィードバックは、短期および長期のメモリに書き込まれます。

8. 最終的な決定と説明を策定します

- LLM は、クエリとツールの出力に基づいて結果を合成するか、アクションを推奨します。

9. レスポンスを返します。

- エージェントは、インターフェイスに結果 (完了したタスク、確認、生成されたコンテンツなど) を返します。

機能

- ビジュアル入力とテキスト入力によるマルチモーダル推論
- シミュレートされた入力または API 駆動型入力によるアプリケーションの制御
- 永続状態のメモリ管理
- シーケンス実行の自律性 (複数ステップフロー)

一般的なユースケース

- IDEs でコードを記述して実行する AI 開発者
- 反復的なデジタルワークフロー用のコンピュータ使用エージェント
- ソフトウェアテストと品質保証のためのシミュレートされたユーザー
- 音声または高レベルの手順で UIs を移動するためのアクセシビリティエージェント
- 推論で強化されたスマートロボットプロセスオートメーション (RPA)

実装のガイド

- このパターンは、以下を使用して構築できます AWS のサービス。
- LLM ベースの計画と推論のための Amazon Bedrock
- シミュレートされた UI 環境でツールサーバーを実行する Amazon Elastic Compute Cloud (Amazon EC2) AWS Lambda、または Amazon SageMaker ノートブック
- メモリ永続化のための Amazon Simple Storage Service (Amazon S3) または Amazon DynamoDB
- ハイブリッドシナリオでの UI イメージ分析用の Amazon Rekognition (またはカスタムモデル)
- オブザーバビリティと監査証跡 AWS X-Ray のための Amazon CloudWatch Logs または

概要

コンピュータを使用するエージェントは自律的なデジタルオペレーターとして機能し、人間とコンピュータのやり取りと AI 主導のアクションの間のギャップを埋めます。メモリ、ツールオーケストレーション、および VLMs を組み込むことで、これらのエージェントは人間向けに設計されたシステムと適応的にやり取りしたり、アクションを実行したり、ファイルを更新したり、メニューを操作したり、レスポンスを生成したりできます。

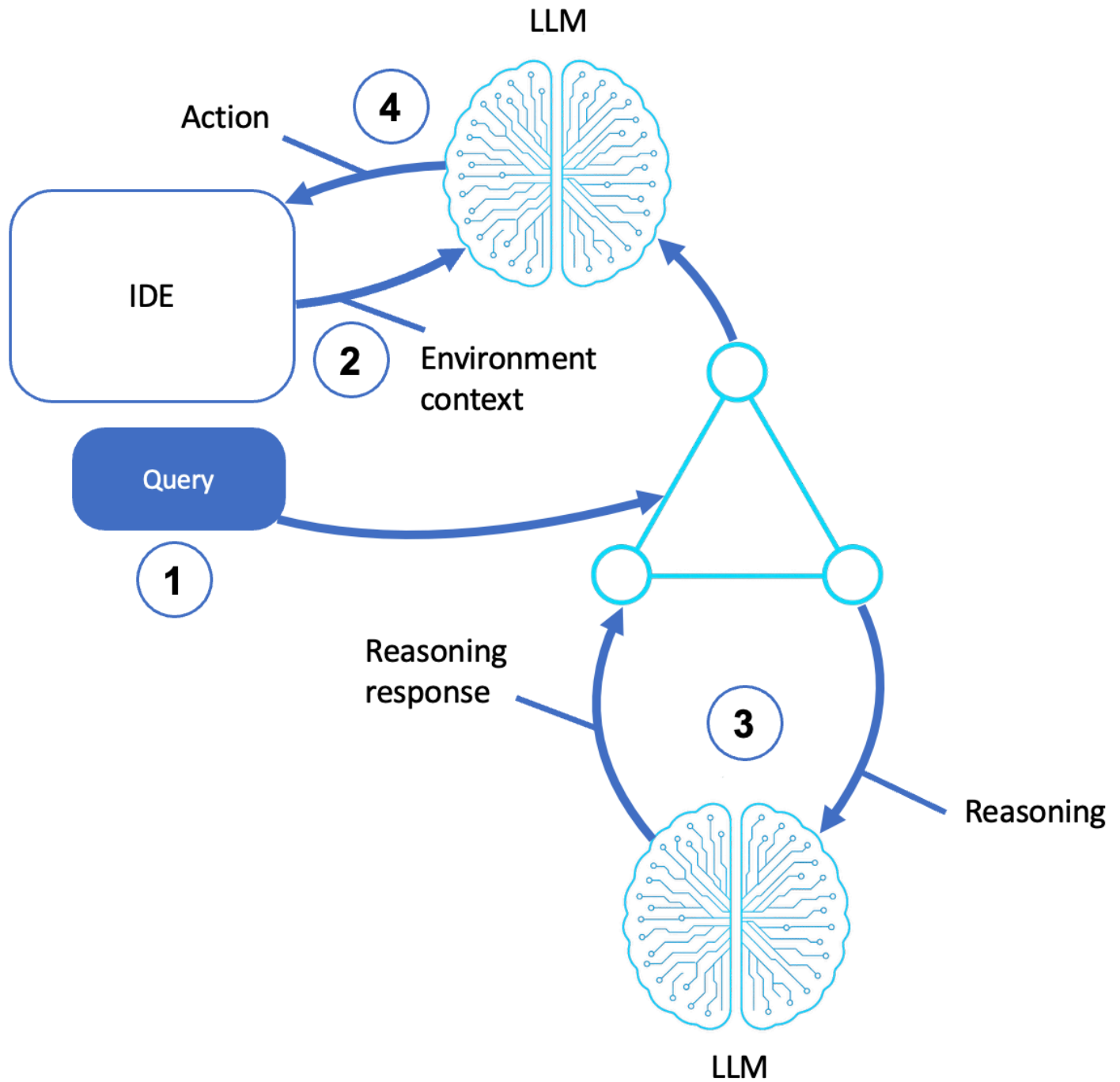
コーディングエージェント

コーディングエージェントは、タスクのプログラミング、コードの生成や変更、IDEs や CLIs。これらのエージェントは、自然言語の理解と構造化された推論を組み合わせ、関数の生成からバグ修正、テストオーサリングまで、ソフトウェア開発を支援、強化、自動化します。

オートコンプリートツールとは異なり、コーディングエージェントはユーザーの目標を積極的に解釈し、開発環境のコンテキストをクエリし (ファイルを開いてエラーを追跡するなど)、要件を特定し、アクションを提案して実行します。

アーキテクチャ

コーディングエージェントのパターンを次の図に示します。



説明

1. クエリを受信する

- ユーザーは、コマンドパレット、チャットウィンドウ、または CLI を通じて自然言語の手順を提供します (たとえば、「この関数にログを追加する」または「読みやすいようにリファクタリングする」)。

2. 環境コンテキストを抽出します

- エージェントは、アクティブなファイル、カーソルの位置、コードスニペット、シンボルテーブルなど、IDE からコンテキストを収集します。
- エラーメッセージ、テスト結果、および他のエージェントからの出力を出力します。

3. LLM 推論

- エージェントは、クエリや環境コンテキストを含むプロンプトを LLM に送信します。
 - LLM は推論パスを実行して以下を決定します。
 - 変更する必要があるもの
 - ソリューションを生成する方法
 - リファクタリング、書き換え、またはコーディングのステップ

4. アクションを実行します

- LLM は出力をエージェントに返し、IDE またはランタイム環境にインポートします。
- これには、コードの挿入または変更、コメントやドキュメントの生成、ダウンストリームのビルド、テスト、リンティングタスクのトリガーなどが含まれます。

機能

- 高コンテキスト対応 (IDE 状態、カーソル、構文ツリーなど)
- 目標とフィードバックの反復的な推論
- オプションのコード計画とアクションの分離 (例: 最初の理由とアクション)
- 同期または非同期の開発者ワークフローで動作

一般的なユースケース

- タスクの説明からのコード生成
- コードのリファクタリングと最適化
- テストケースの生成と検証
- エラーの説明とデバッグ
- ドキュメントアシスタント
- ペアプログラミングコパイロット

実装のガイド

- このパターンは、次のツールとを使用して構築できます AWS のサービス。
- LLM 駆動型の生成と推論のための Amazon Bedrock
- コーディングの提案と完了のための Amazon Q Developer
- AWS Lambda サンドボックス環境を実行およびテストするための または Amazon Elastic Container Service (Amazon ECS)
- AWS Cloud9、VS Code 拡張機能、またはコンテキストをホストおよび評価するためのカスタム IDE 統合
- 中間プロンプト、レスポンス、リビジョン履歴を保存するための Amazon Simple Storage Service (Amazon S3)

概要

コーディングエージェントは、自然言語の解釈、コンテキストの分析、複数ステップのコード変更の生成、ソフトウェア開発ライフサイクルとの統合が可能な新しい AI を活用した開発ツールです。

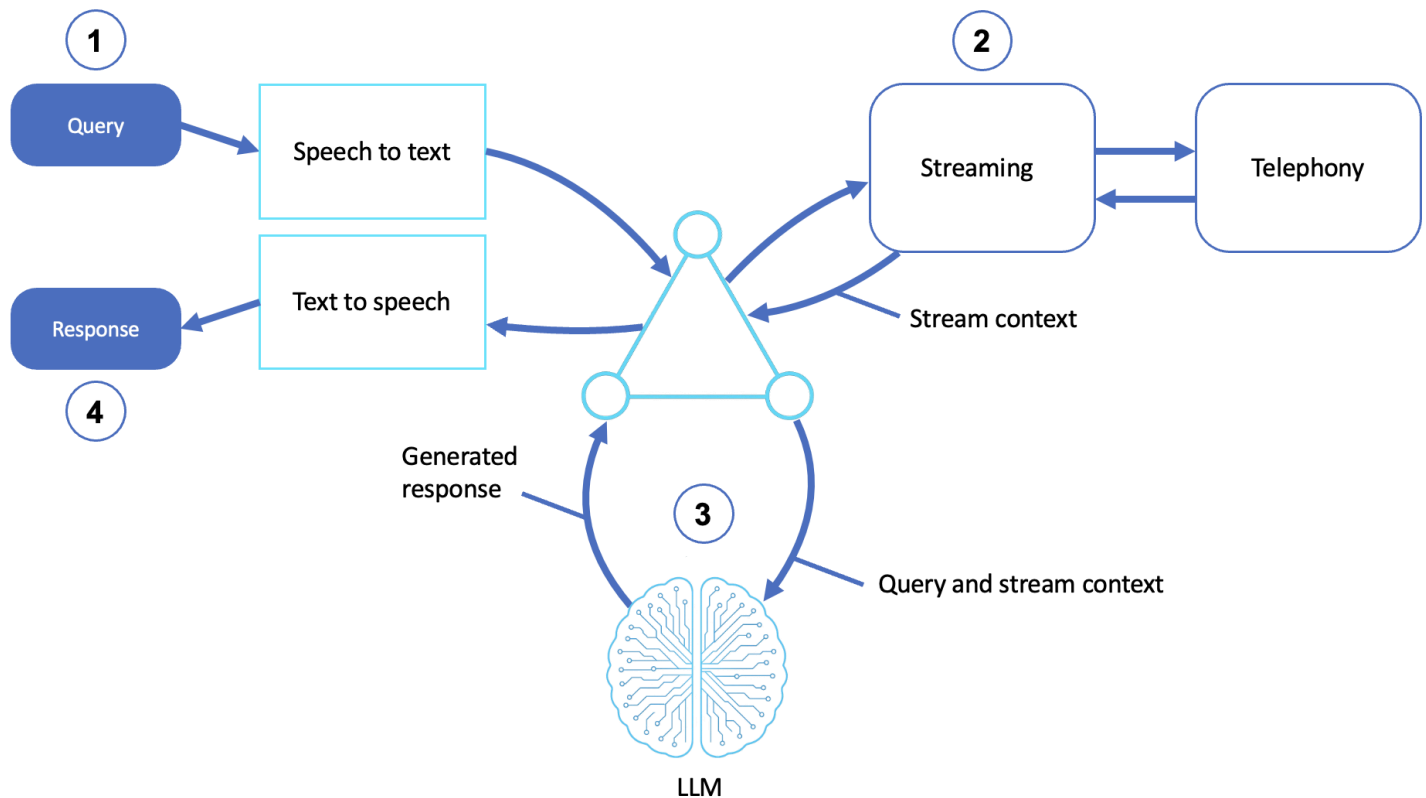
音声エージェント

音声エージェントと音声エージェントは、会話を通じてユーザーとやり取りします。これらのエージェントは、音声認識、自然言語理解、音声合成を統合して、テレフォニー、モバイル、ウェブ、埋め込みプラットフォーム全体で会話 AI を可能にします。

音声エージェントは、ハンドフリー、リアルタイム、またはアクセシビリティ駆動型の環境で特に効果的です。ストリーミングインターフェイスと LLM を活用した推論を組み合わせることで、ユーザーにとって自然なリッチで動的なインタラクションが容易になります。

アーキテクチャ

音声エージェントと音声エージェントを次の図に示します。



説明

1. 音声クエリを受信する

- ユーザーは電話、マイク、または埋め込みシステムにリクエストを発声します。
- speech-to-text (STT) モジュールは、音声をテキストに変換します。

2. ストリーミングコンテキストとテレフォニーコンテキストを統合する

- エージェントはストリーミングインターフェイスを使用して、オーディオ I/O をリアルタイムで管理します。
- コンタクトセンターまたはテレコムコンテキストにデプロイされている場合、テレフォニー統合はセッションルーティング、デュアルトーンマルチ周波数 (DTMF) 入力、およびメディアトランスポートを処理します。

注: DTMF は、電話キーパッドのボタンを押すと生成されるトーンを指します。音声エージェント内でのストリーミングおよびテレフォニーコンテキスト統合のコンテキストでは、DTMF は、特にインタラクティブ音声応答 (IVR) システムにおいて、通話中の信号入力メカニズムとして使用されます。DTMF 入力により、エージェントは次のことを実行できます。

- メニューの選択を認識します (たとえば、「請求の場合は 1 を押します。サポートの場合は 2 を押します。」)
- 数値入力 (アカウント番号、PINs 収集する)
- コールフローのワークフローまたは状態遷移をトリガーする
- 必要に応じて音声からタッチトーンに戻る

1. LLM ストリームコンテキストによる理由

- クエリはエージェントに送信され、セッションメタデータ (発信者 ID、以前のコンテキストなど) とともに LLM に渡されます。
- LLM は、インタラクションが進行中の場合は、chain-of-thought 戦略またはマルチターンメモリを使用してレスポンスを生成します。

2. 音声レスポンスを返します。

- エージェントは、テキスト読み上げ (TTS) を使用して text-to-speech に変換します。
- 音声チャンネルを介してユーザーに音声を返します。

機能

- リアルタイムの音声の理解と生成
- STT と TTS をサポートする多言語 I/O
- テレフォニー API またはストリーミング APIs との統合
- ターン間のセッション認識とメモリのハンドオフ

一般的なユースケース

- 対話型 IVR システム
- 仮想受付担当者と予約スケジューラ
- 音声駆動型ヘルプデスクエージェント
- ウェアラブル音声アシスタント
- スマートホームとアクセシビリティツール用の音声インターフェイス

実装のガイド

このパターンは、次のツールとを使用して構築できます AWS のサービス。

- STT 用の Amazon Lex V2 または Amazon Transcribe
- TTS 用の Amazon Polly
- ストリーミングとテレフォニー用の Amazon Chime SDK、Amazon Connect Customer、または Amazon Interactive Video Service (Amazon IVS)
- Anthropic、AI21、またはその他の基盤モデルによる推論のための Amazon Bedrock
- AWS Lambda STT、LLM、TTS、およびセッションコンテキストを接続するには

(オプション) 追加の機能強化には以下が含まれます。

- Amazon Kendra または OpenSearch for コンテキスト対応 RAG
- セッションメモリ用の Amazon DynamoDB
- トレーサビリティ AWS X-Ray のための Amazon CloudWatch Logs と

概要

音声エージェントは、自然な会話を通じてやり取りするインテリジェントなシステムです。音声インターフェイスを LLM 推論およびリアルタイムストリーミングインフラストラクチャと統合することで、音声エージェントはシームレスでアクセス可能でスケーラブルなインタラクションを可能にします。

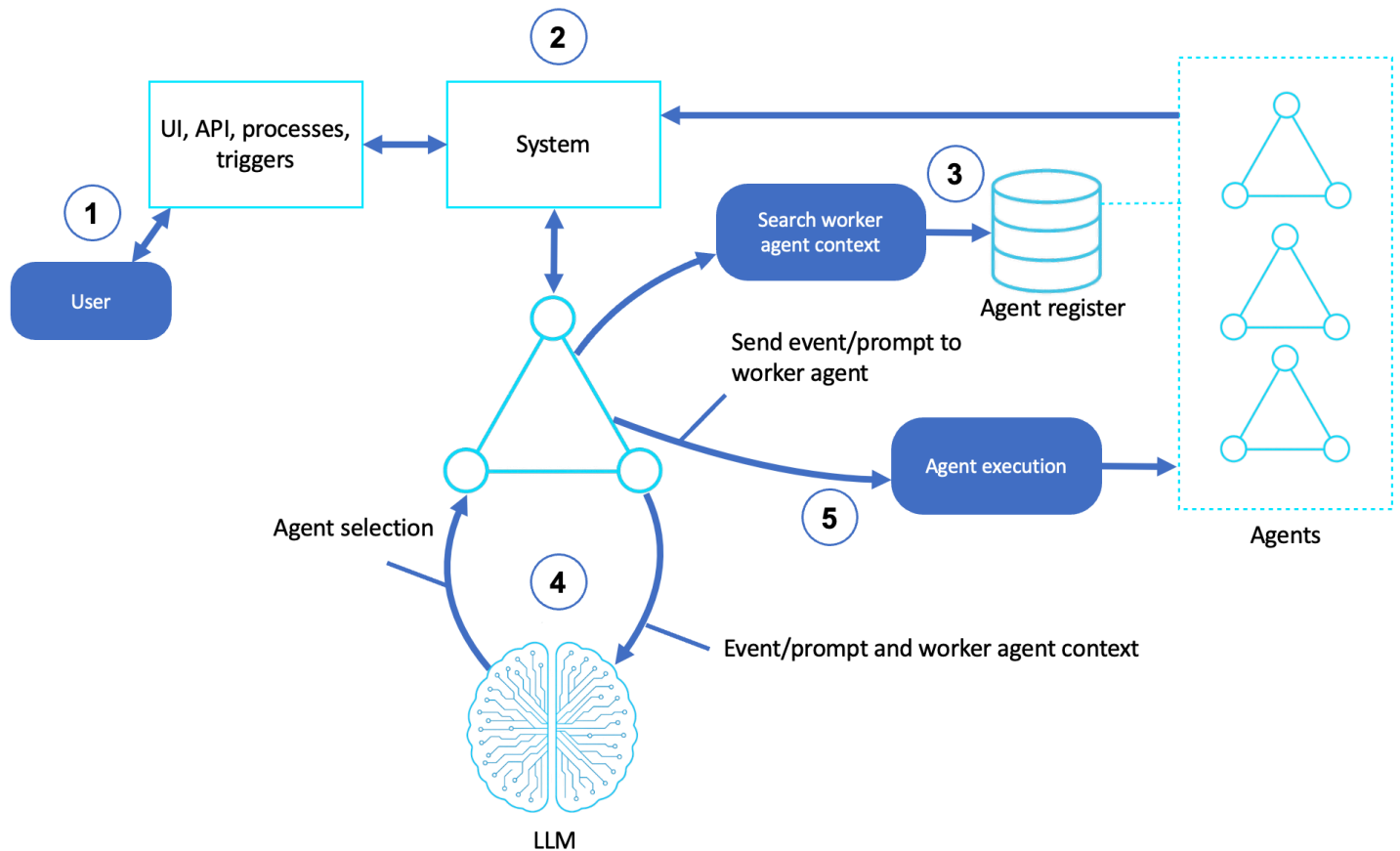
ワークフローオーケストレーションエージェント

ワークフローオーケストレーションエージェントは、分散システム全体の複数ステップのタスク、プロセス、サービスを管理および調整します。これらのエージェントは、推論や単独での行動ではなく、作業をサブエージェントや他のシステムに委任し、実行コンテキストを維持し、中間結果に基づいて適応します。

これらのエージェントは自動化フローの基本的な部分です。これらは、さまざまなエージェントやツールを順番または条件付きで呼び出す必要がある、長時間実行されるタスク、マルチエージェント構成、クロスドメイン統合を処理する場合に特に便利です。

アーキテクチャ

ワークフローオーケストレーションエージェントを次の図に示します。



説明

1. ユーザー入力を受信します

- ユーザー (または外部トリガー) は、UI、API、またはシステムイベントを通じてタスクを開始します。

2. システムイベントを処理します

- システムコンポーネントはリクエストを受け取り、オーケストレーションを必要とするイベントまたはコマンドを出力します。

3. コンテキストを取得します。

- ワークフローエージェントは、ナレッジベースとエージェントレジストリをクエリして、メタデータ、ドメイン、以前の成功率に基づいてタスクに適したワーカーエージェントを見つけます。

4. LLM エージェントを選択します。

- LLM は、タスクの説明と使用可能なオプションを分析することで、最適なエージェントまたはワークフロープランを選択するのに役立ちます。
- また、タスク固有のプロンプトを作成して、選択したエージェントに送信することもできます。

5. 委任と実行

- 選択したワーカーエージェントはイベントまたはプロンプトを受け取り、コマンドの実行を開始します。
- 実行状態を追跡し、失敗時に再試行し、シーケンス内の次のエージェントに中間結果を渡すことができます。

機能

- エージェント構成 (スーパーバイザー、共同作業エージェント、ツールなど)
- イベント駆動型またはスケジュールされた実行
- 時間の経過に伴うメモリと状態の追跡
- 階層型または並列タスクのオーケストレーション (非同期ワークフローと比較した同期)
- 動的なエージェントの選択と連鎖

一般的なユースケース

- マルチステップオートメーション (データの取り込みやレポートなど)
- カスタマーサービスのルーティングとエスカレーション (agent-as-coordinatorなど)
- AI エージェントは、同じループ内の人間やボットと連携します。
- LLM を活用したロジックを使用してエンタープライズプロセスを自動化する
- ハイブリッドシステムは AI エージェントと従来のオーケストレーションツールを組み合わせる

実装のガイド

このパターンは、次のツールとを使用して構築できます AWS のサービス。

- 推論とエージェント選択のための Amazon Bedrock
- AWS Step Functions ワークフロー構成用の または Amazon EventBridge
- AWS Lambda 実行ユニットまたはタスクランナーとして

- 状態と結果を追跡するための Amazon DynamoDB、Amazon Simple Storage Service (Amazon S3)、または Amazon RDS
- AWS AppFabric または Amazon AppFlow によるクロスシステム調整
- (オプション) Amazon SageMaker 実行エージェントを使用してドメイン固有のワーカーエージェントをホストする

概要

ワークフローエージェントは、マルチエージェント環境で目標を調整、適応、調整します。つまり、AI エージェントは、モジュール式の説明可能なワークフローを通じて、コラボレーション、ランタイム条件への適応、複雑な結果の提供を行うことができます。

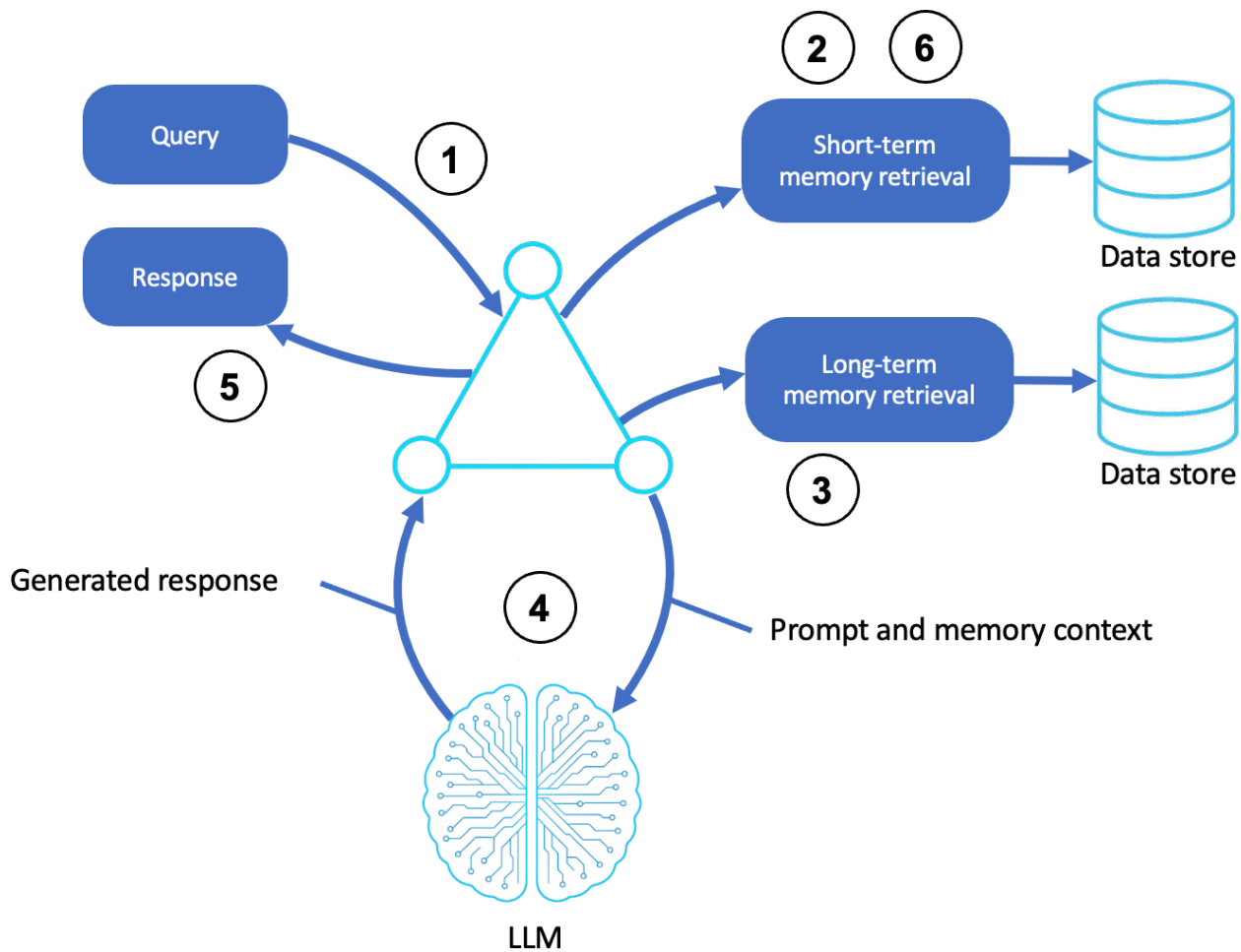
メモリ拡張エージェント

メモリ拡張エージェントは、短期メモリと長期メモリを使用して、保存、取得、理由の機能が強化されています。これにより、複数のタスク、セッション、インタラクションにわたってコンテキストを維持できるため、より一貫性があり、パーソナライズされ、戦略的に対応することができます。

ステートレスエージェントとは異なり、メモリ拡張エージェントは履歴データを参照して適応し、以前の結果から学び、ユーザーの目標、好み、環境に合った意思決定を行います。

アーキテクチャ

メモリ拡張エージェントを次の図に示します。



説明

1. 入力またはイベントを受信します

- エージェントはユーザークエリまたはシステムイベントを受け取ります。これは、テキスト、API トリガー、または環境変更である可能性があります。

2. 短期メモリを取得します。

- エージェントは、セッションまたはワークフローに関連する最近の会話履歴、タスクコンテキスト、またはシステム状態を取得します。

3. 長期メモリを取得します

- エージェントは、次のような履歴インサイトについて長期メモリ (ベクトルデータベースやキーバリューストアなど) をクエリします。
 - ユーザー設定
 - 過去の決定と結果

- 学習した概念、概要、経験

4. LLM 経由の理由

- メモリコンテキストは LLM プロンプトに埋め込まれているため、エージェントは現在の入力と以前の知識の両方に基づいて推論できます。

5. 出力を生成します

- エージェントは、タスク履歴とユーザーの入力に応じてパーソナライズされた、コンテキストに応じた対応、計画、またはアクションを生成します。

6. メモリの更新

- 更新された目標、成功と失敗のシグナル、構造化されたレスポンスなどの新しい情報は、将来のタスクのために保存されます。

機能

- 会話またはイベント間のセッション継続性
- 時間の経過に伴う目標の永続性
- 進化する状態に基づくコンテキスト認識
- 以前の成功と失敗から得られる適応性
- ユーザー設定と履歴に合わせたパーソナライゼーション

一般的なユースケース

- ユーザー設定を記憶する会話型副操縦士
- コードベースの変更を追跡するコーディングエージェント
- タスク履歴に従って適応するワークフローエージェント
- システム知識から進化するデジタルツイン
- 冗長な取り出しを回避する調査エージェント

メモリ拡張エージェントの実装

メモリ拡張エージェント AWS のサービス には、次のツールと を使用します。

メモリレイヤー	AWS のサービス	目的
---------	-----------	----

短期	Amazon DynamoDB、 Redis、Amazon Bedrock コン テキスト	最近のインタラクション状態 の迅速な取得
長期 (構造化)	Amazon Aurora、Amazon DynamoDB、Amazon Neptune	事実、関係、ログ
長期 (セマンティック)	OpenSearch、Postgre SQL、Pinecone	埋め込みベースの取得 (RAG)
Storage	Amazon S3	トランスクリプト、構造化メ モリ、ファイルの保存
オーケストレーション	AWS Lambda or AWS Step Functions	メモリインジェクションと更 新ライフサイクルの管理
推論	Amazon Bedrock	Anthropic Claude または Mistral とメモリプロンプト

メモリ注入プロンプトの実装

メモリをエージェントの推論に統合するには、構造化状態と取得拡張コンテキストインジェクションの組み合わせを使用します。

- 言語モデルのプロンプトを構築するときは、最新のエージェント状態と最近の対話履歴を構造化された入力として含めます。これにより、完全なコンテキストで推論できます。
- 検索拡張生成 (RAG) を使用して、関連するドキュメントや事実を長期メモリから取得します。
- 圧縮と関連性に関する以前の計画、コンテキスト、インタラクションを要約します。
- 推論中にベクトルストアや構造化ログなどの外部メモリモジュールを挿入して、意思決定をガイドします。

概要

メモリ拡張エージェントは、経験から学び、ユーザーコンテキストを記憶することで、思考の継続性を維持します。これらのエージェントは、長期的なコラボレーション、パーソナライゼーション、戦略的推論を使用することで、リアクティブインテリジェンスを上回ります。エージェント AI に関し

て、メモリを使用すると、エージェントはアダプティブデジタルのように動作し、ステートレスツールのように動作しなくなります。

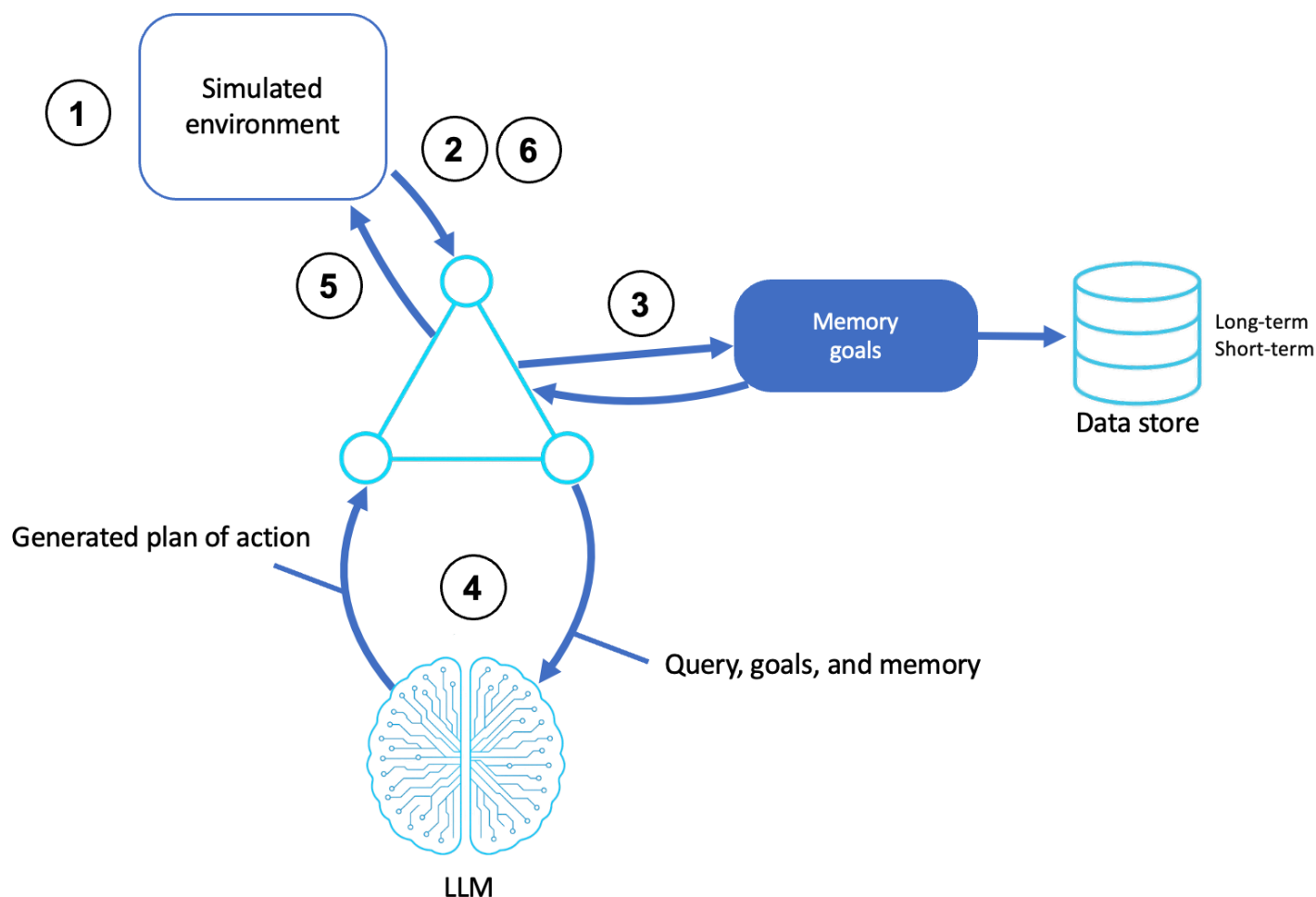
シミュレーションおよびテスト床エージェント

シミュレーションエージェントとテスト床エージェントは、理由、行動、学習を行う仮想化環境または制御環境内で動作します。これらのエージェントは、実際の環境に適用する前に、動作をシミュレートし、結果をモデル化し、繰り返し可能な設定で戦略をトレーニングします。

このパターンは、反復開発、強化学習 (RL)、自律的な意思決定評価、および緊急行動テストに役立ちます。シミュレーションエージェントは多くの場合、クローズドループで動作し、環境からフィードバックを受け取り、それに応じて動作を調整するため、空間推論、リアルタイム制御、複雑なシステムダイナミクスを含むタスクにとって不可欠です。

アーキテクチャ

次の図は、シミュレーションエージェントまたはテスト床エージェントを示しています。



説明

1. 環境を開始する

- エージェントはシミュレートされた環境 (3D ワールド、物理エンジン、CLI サンドボックス、合成データストリームなど) を開始します。
- エージェントは、初期タスク、目標、またはポリシーを使用して環境にロードされます。

2. エージェントを認識する

- エージェントはシミュレーションテレメトリ (センサーエミュレーション、仮想カメラ、構造化ログなど) を通じて現在の状態を認識します。

3. 目標とメモリを取得します。

- エージェントは、割り当てられた目標、シナリオ手順、またはコンテキスト目標を取得します。
- また、以下を含む以前のメモリを取得することもできます。

- 長期戦略またはポリシー
- 環境マップまたは既知の制約
- 同様のシミュレーションによる過去の成功または失敗

4. 理由と計画

- LLM は、シミュレートされた状態、タスクの目的、学習した知識を解釈します。
- アクションプランまたはコントロールコマンドを生成します。

5. シミュレートされたアクションを実行します

- エージェントは、プランの実行、状態の変更、スペースのナビゲート、仮想エンティティとのやり取りを行います。

6. 学習

- エージェントはアクションの結果を評価します
- エージェントの設定によっては、以下を行う場合があります。
 - RL を実行する
 - 将来のファインチューニングのログ結果
 - リアルタイムで戦略を適応させる

機能

- 合成環境または仮想環境内で動作します
- trial-and-error学習、ポリシーの改良、システムモデリングをサポート
- 動作、障害処理、エッジケースの低リスクテスト
- マルチエージェント設定で緊急エージェント動作分析を有効にする
- クローズドループ制御とhuman-in-the-loop探索の両方をサポート

一般的なユースケース

- ロボット、ドローン、ゲームの強化学習
- 仮想道路での自律車両トレーニング
- DevOps およびテスト床シナリオ用のシミュレートされた UIs または CLIs
- ソーシャルシミュレーションにおける緊急の動作実験
- 本番稼働前の決定ロジックの安全性検証

実装のガイド

次のツールとを使用して、シミュレーションおよびテスト床エージェントを構築できます AWS のサービス。

コンポーネント	AWS のサービス	目的
環境	Amazon SageMaker スタジオ ラボの Amazon ECS、Amazon Amazon EC2、またはカスタ ムシミュレーター	仮想ワールド (Gazebo、U nity、Unreal) またはサンド ボックス CLIs を実行する
エージェントロジック	Amazon Bedrock、Amazon SageMaker、または AWS Lambda	LLM ベースのプランナーまた は RL エージェント
フィードバックループ	Amazon SageMaker 強化学 習、Amazon CloudWatch、ま たはカスタムログ	報酬の追跡、結果のスコアリ ング、動作のログ記録
メモリとリプレイ	Amazon S3、Amazon DynamoDB、または Amazon RDS	永続的な状態、エピソード履 歴、またはシナリオデータ
視覚的表現	Amazon CloudWatch ダッ シュボードまたは Amazon SageMaker ノートブック	ポリシーの変更、結果、ト レーニングメトリクスを確認 する

追加のアプリケーションは次のとおりです。

- [AWS SimSpace Weaver](#) 大規模な空間シミュレーション用
- [AWS IoT Core](#) シャドウデバイスをテストするための
- エージェントの評価とベンチマークのための [Amazon SageMaker Experiments](#)

概要

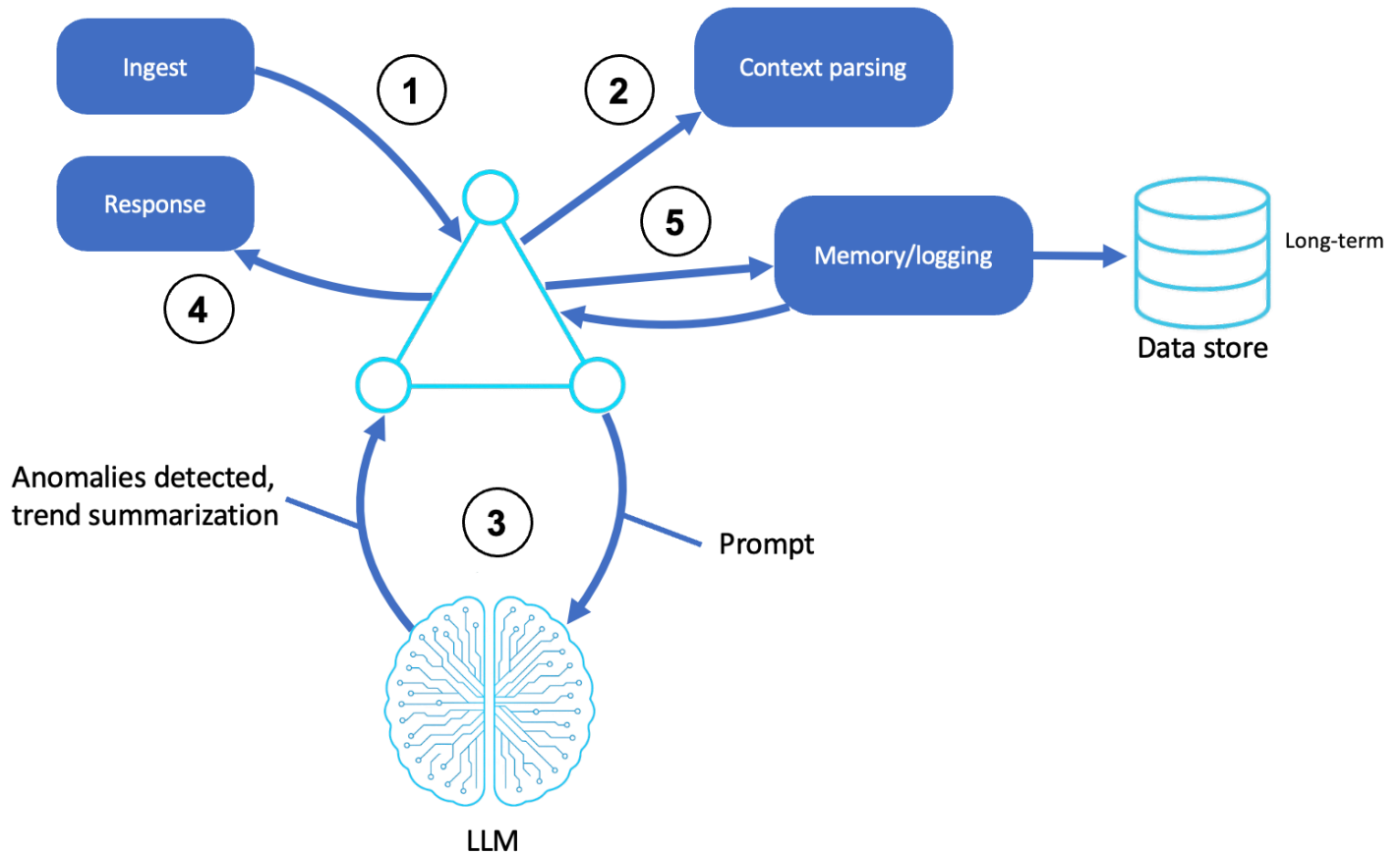
シミュレーションエージェントとテスト床エージェントは、実稼働システムにデプロイされる前に構造化された探索用です。これらのエージェントを使用して、自律ナビゲーションポリシーのトレーニング、合成環境でのビジネスプロセスのテスト、調整パターンのスワームの評価を行います。

オブザーバーエージェントとモニタリングエージェント

オブザーバーエージェントとモニタリングエージェントは、システム、環境、インタラクションをパッシブに監視して、パターンの検出、インサイトの生成、アクションのトリガーを行います。インテリジェントな監視者として、動作を直接開始することなく、アラート、診断、監査を強化します。

これらのエージェントは、従来のモニタリングに適応性や推論がない場合、特に AI-in-the-loop モニタリング、異常検出、コンプライアンス監視、セキュリティインテリジェンスに優れています。オブザーバーエージェントは、システムテレメトリとユーザーインタラクションを継続的にモニタリングするイベントリスナーです。エージェントは、認識、解釈、および条件付きエスカレーションまたは報告に依存します。

アーキテクチャ



説明

1. テレメトリの取り込み

- エージェントは、次のような 1 つ以上のシステムソースから入力を受け取ります。
 - ログ (アプリケーション、インフラストラクチャ、セキュリティ)
 - メトリクス (パフォーマンス、レイテンシー、使用状況)
 - イベント (API コール、ユーザーアクション、センサーデータ)

2. コンテキストを解析する

- Raw 入力は、タイムスタンプ、アクターアイデンティティ、システム状態、トレース ID などのメタデータで解析、構造化、強化されます。

3. LLM を使用する理由

- エージェントは LLM またはロジックモジュールを使用して、異常を特定し、傾向を要約し、分散トレースまたは時間枠間で関連させることで、解析された入力を解釈します。

4. 分類またはアラート

- エージェントは、観察された動作が以下を必要とするかどうかを判断します。
 - アラートまたはエスカレーション
 - レポートまたはダッシュボードの更新
 - レスポンストリガー (自動修復やポリシーの適用など)

5. ログメモリまたはフィードバックループ

- エージェントは、長期学習、監査、または他のエージェントの将来の参照に関するイベントと決定を保存します。

機能

- パッシブおよび非侵襲的 (エージェントは直接動作しません)
- 高度にスケーラブルで非同期
- ノイズの多いシグナルまたは分散シグナル間の AI 主導の相関関係
- 監査、コンプライアンス、リアルタイムインサイトをサポート
- ダウンストリームエージェントまたは人間のワークフローにフィードできる

一般的なユースケース

- マイクロサービスと APIs の AI 拡張オブザーバビリティ
- モデルドリフト、ポリシー違反、帯域out-of-band動作のモニタリング
- 顧客アクティビティ分析またはインタラクションの概要
- コミットまたはデプロイをモニタリングするコードレビューエージェント
- LLM 推論を使用したセキュリティまたはコンプライアンスログのモニタリング

実装のガイド

オブザーバーとモニタリングエージェントは、次のツールとを使用して構築できます AWS のサービス。

コンポーネント	AWS のサービス	目的
イベントの取り込み	Amazon EventBridge、Amazon CloudWatch	構造化テレメトリと非構造化テレメトリを取り込む

	Logs、Amazon Kinesis、Amazon S3	
前処理	AWS Lambda、AWS Glue、AWS Step Functions	raw データを構造化プロンプトに変換する
推論エンジン	Amazon Bedrock、Amazon SageMaker、AWS Lambda	イベントの分析、動作の分類、インサイトの生成
ストレージとメモリ	Amazon S3、Amazon DynamoDB、OpenSearch	永続的な観測値、概要、出力
アラートとエスカレーション	Amazon SNS、AWS AppFabric、Amazon EventBridge	ダウンストリームシステムまたはエージェントのトリガー

追加のアプリケーションは次のとおりです。

- [AWS Security Hub CSPM](#) セキュリティログモニタリング用
- エージェント出力を視覚化するための [Amazon Quick](#)

概要

オブザーバーエージェントとモニタリングエージェントは、システムや動作をリアルタイムで追跡します。人間やルールが見逃す可能性のあるパターンを特定することで、異常を検出し、セキュリティを監査し、運用インテリジェンスを収集します。この機能は、変化する条件に適応し、包括的なデータ分析に基づいて意思決定できるシステムを作成するのに役立ちます。

マルチエージェントコラボレーション

マルチエージェントコラボレーションとは、それぞれが異なるロール、専門分野、または目標を持つ複数の自律型エージェントが、複雑なタスクを解決するために交渉するパターンを指します。これらのエージェントは、情報を共有し、責任を分割し、目標に向かって集合的に推論することで、独立して、または他のエージェントと運用できます。

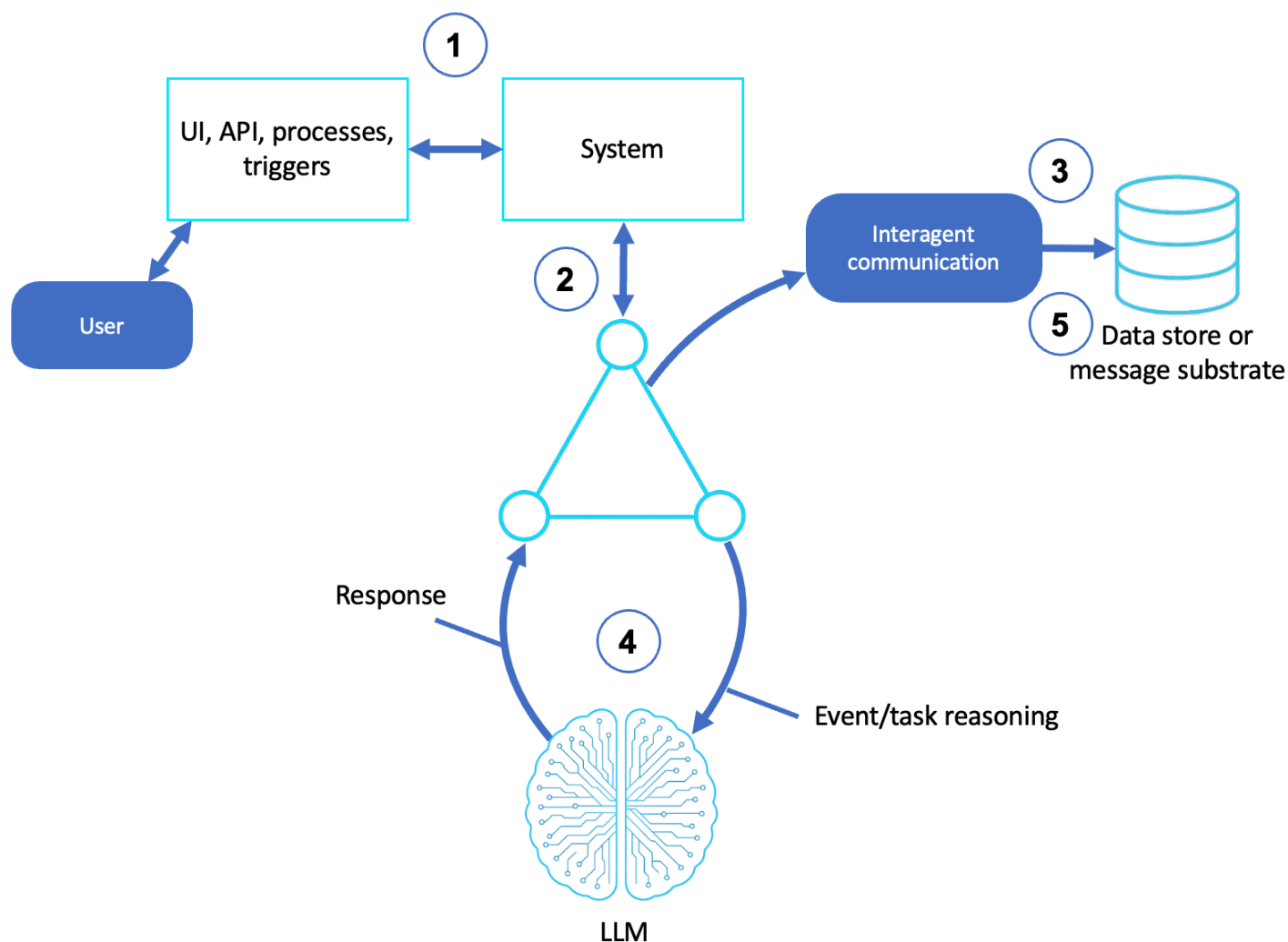
このパターンは、構造化フロー内の下位エージェントにタスクを一元的に調整して委任するワークフローエージェントとは異なります。対照的に、マルチエージェントコラボレーションは、適応性、並

列性、認識の分割を可能にすることで、peer-to-peerまたは緊急の調整を強調します。次の表は、マルチエージェントコラボレーションとワークフローエージェントを比較したものです。

機能	ワークフローエージェント	目的
コントロール	一元化されたコーディネーター	分散型、分散型、またはロールベースのピア
相互作用	1 人のエージェントが実行を委任して追跡する	複数のエージェントが交渉、共有、適応する
設計	事前定義されたタスクのシーケンス	エマージェントで柔軟なタスクディストリビューション
調整	手続き型オーケストレーション	協力的または競争的なやり取り
ユースケース	エンタープライズプロセスの自動化	複雑な推論、探索、緊急戦略

アーキテクチャ

次の図は、マルチエージェントコラボレーションを示しています。



説明

1. タスクを開始する

- ユーザーまたはシステムは、高レベルの目標または問題を発行します。
- 「マネージャー」エージェントまたは開始コンテキストは、目標を定義します。

2. ロールの割り当てまたは検出

- エージェントは、プランナー、研究者、エグゼキュター、批評家、説明者などの他のロールに自己割り当て (シンボリックロジックまたは推論) または委任 (イベントブローカー) されます。

3. 他のエージェントと通信する

- エージェントは、共有メモリ、メッセージングキュー、またはプロンプトチェーンを介して通信します。
- 相互にサブタスクの議論、クエリ、提案を行う場合があります。

4. 特殊な推論を使用する

- 各エージェントは、独自のモデルまたはドメインロジックを使用して、問題の一部を解決します。
- エージェントは、ロール固有のプロンプトとメモリで LLMs を使用できます。

5. 出力または目標を調整する

- エージェントは、投稿を最終的な回答、計画、またはアクションに合成します。
- (オプション) 監視エージェントは、合成された出力を検証または要約できます。

機能

- 特殊なロールまたはスキルを持つピアレベルのエージェント
- コミュニケーションまたはネゴシエーションによる緊急の動作
- 複雑または多面的な問題の並列処理
- 熟考、自己修正、リフレクションの反復をサポート
- ソーシャルダイナミクス、科学コラボレーション、またはエンタープライズチームの役割をモデル化する

一般的なユースケース

- 自律型研究チーム (検索エージェント、要約者、検証者)
- ソフトウェア開発 (プランナー、コーダー、テスター)
- ビジネスシナリオモデリング (財務、ポリシー、コンプライアンス)
- 交渉、入札、またはマルチパーティーの推論
- マルチモーダルタスク (イメージ、テキスト、ロジック)

実装のガイド

次のツールとを使用して、マルチエージェントシステムを構築できます AWS のサービス。

コンポーネント	AWS のサービス	目的
エージェントホスティング	Amazon Bedrock、Amazon SageMaker、AWS Lambda	個々の LLM 駆動型エージェントをホストする

通信レイヤー	Amazon SQS、Amazon EventBridge、AWS AppFabric	エージェント間のメッセージングと調整
共有メモリ	Amazon DynamoDB、Amazon S3、または OpenSearch	マルチエージェントメモリまたはブラックボードシステム
オーケストレーションレイヤー	AWS Step Functions、AWS Lambda パイプライン	キックオフ、タイムアウト、フォールバック、再試行ロジック
エージェント ID	Amazon Bedrock エージェント (ロール定義) AWS AppConfig、および Amazon Bedrock converse API (Amazon Bedrock 外のエージェント)	ロールベースのツールまたはエージェントの呼び出しと境界の適用
緊急時のやり取り	Amazon EventBridge パイプラインまたはエージェントレジストリ	動的タスクルーティングまたはエスカレーションを有効にする

概要

マルチエージェントコラボレーションは、問題解決タスクをモジュール式のロール駆動型エージェントに分散します。ワークフローオーケストレーションとは異なり、コラボレーションパターンは、人間が問題を解決する方法を反映する緊急のインテリジェンス、レジリエンス、スケーラビリティを使用します。これは、オープンエンドドメイン、クリエイティブタスク、マルチモーダル推論、および多様な視点から恩恵を受ける環境にとって特に重要です。

結論

前述のパターンは、エージェント AI の実際の実装に対する基本的なアプローチを示しています。基本的な推論からメモリ拡張インテリジェンスまで、各パターンは、自律性、非同期性、および機関に基づく認識、認識、およびアクションに対して一意に設定されます。

これらのパターンは、インテリジェントで目標指向のシステムを構築するための語彙と技術的な設計図を共有します。パターンがユーザーインターフェイスに埋め込まれているか、クラウドサービスを

介してオーケストレーションされているか、エージェントのチーム間で調整されているかにかかわらず、各パターンは適応可能でモジュール式です。

重要事項

- エージェントパターンは構成可能 – ほとんどの実世界のエージェントは、2 つ以上のパターン (たとえば、ツールベースの推論とメモリを持つ音声エージェント) をブレンドします。
- エージェント設計はコンテキストに基づく – インタラクションサーフェス、タスクの複雑さ、レイテンシー耐性、ドメイン固有の制約に基づいてパターンを選択します。
- AWS ネイティブ実装を実現可能 – Amazon Bedrock、Amazon SageMaker AWS Lambda AWS Step Functions、およびイベント駆動型アーキテクチャでは、すべてのエージェントパターンを大規模に配信できます。

LLM ワークフロー

エージェントパターンでは、認識、アクション、学習、認識の一連のモジュラー機能を中心に構築された一般的な AI エージェントパターンを調べました。多くのエージェントパターンの認知モジュールの中心には、推論、計画、意思決定が可能な大規模言語モデル (LLM) があります。ただし、LLM のみを呼び出すだけでは、インテリジェントな目標指向の動作を生成するには不十分です。

複雑なタスクを確実に実行するには、エージェントは構造化ワークフロー内に LLM を埋め込む必要があります。構造化ワークフローでは、モデルの機能がツール、メモリ、計画ループ、調整ロジックで強化されます。これらの LLM ワークフローにより、エージェントは目標を分割し、サブタスクをルーティングし、外部サービス呼び出し、結果を反映し、他のエージェントと調整できます。

この章では、再利用可能なワークフローを中心に整理された、堅牢で拡張可能なインテリジェントな LLM 駆動型コグニティブモジュールを構築するためのコア設計パターンについて説明します。

このセクションの内容

- [LLM で拡張された認識の概要](#)
- [プロンプト連鎖のワークフロー](#)
- [ルーティングのワークフロー](#)
- [並列化のワークフロー](#)
- [オーケストレーションのワークフロー](#)
- [評価者とリフレクション/絞り込みループのワークフロー](#)
- [結論](#)

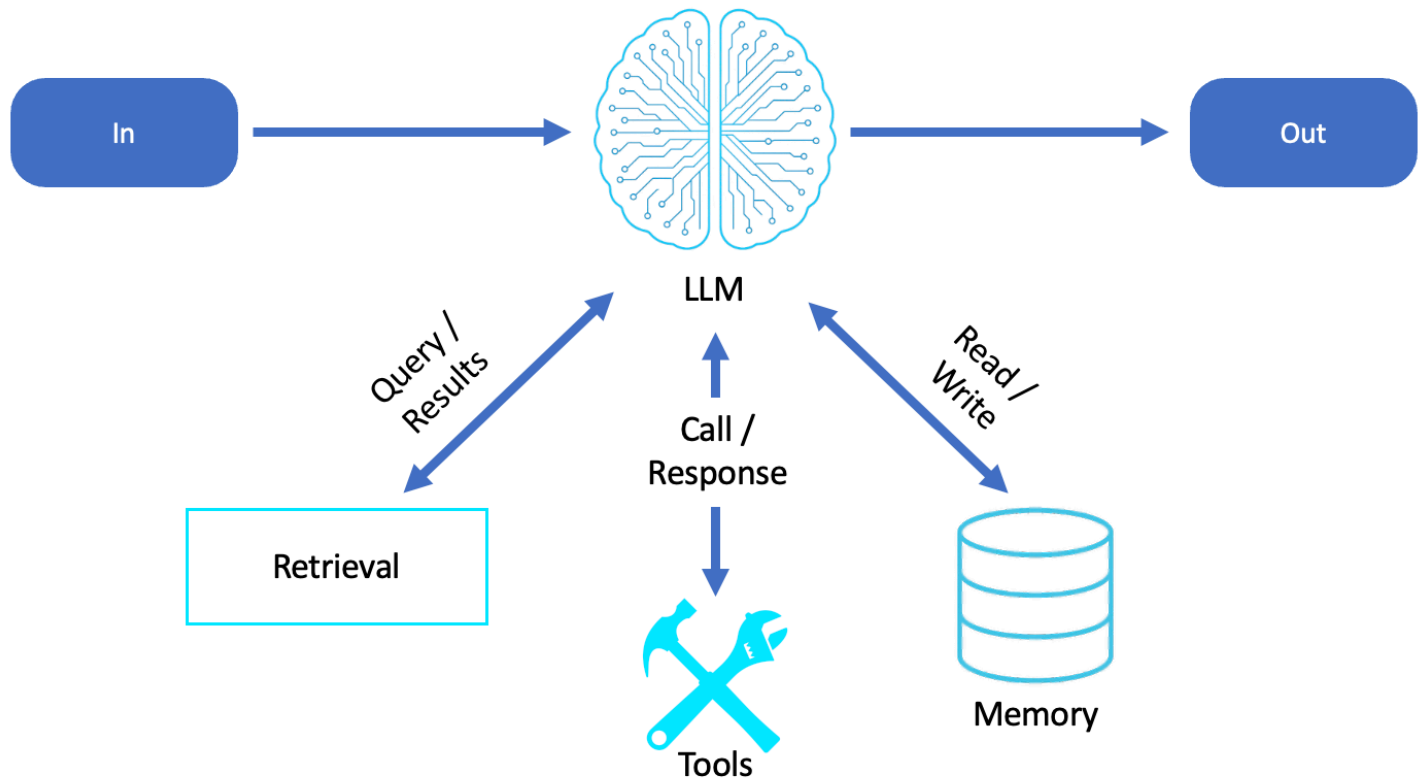
LLM で拡張された認識の概要

その中核として、ソフトウェアエージェントのコグニティブモジュールは、拡張でラップされた LLM として見ることができます。エージェントは、次の構成要素を使用して、環境内で効果的に推論できます。

- プロンプト — コンテキスト、命令、例、メモリを使用した入力のフレーミング
- 取得 — ベクトル検索やセマンティックメモリなど、取得拡張生成 (RAG) を通じて、LLM プロンプトに up-to-date 知識やドメイン固有の知識を提供します。
- ツールの使用 — LLM が APIs が、関数を呼び出して情報を取得または処理できるようにする

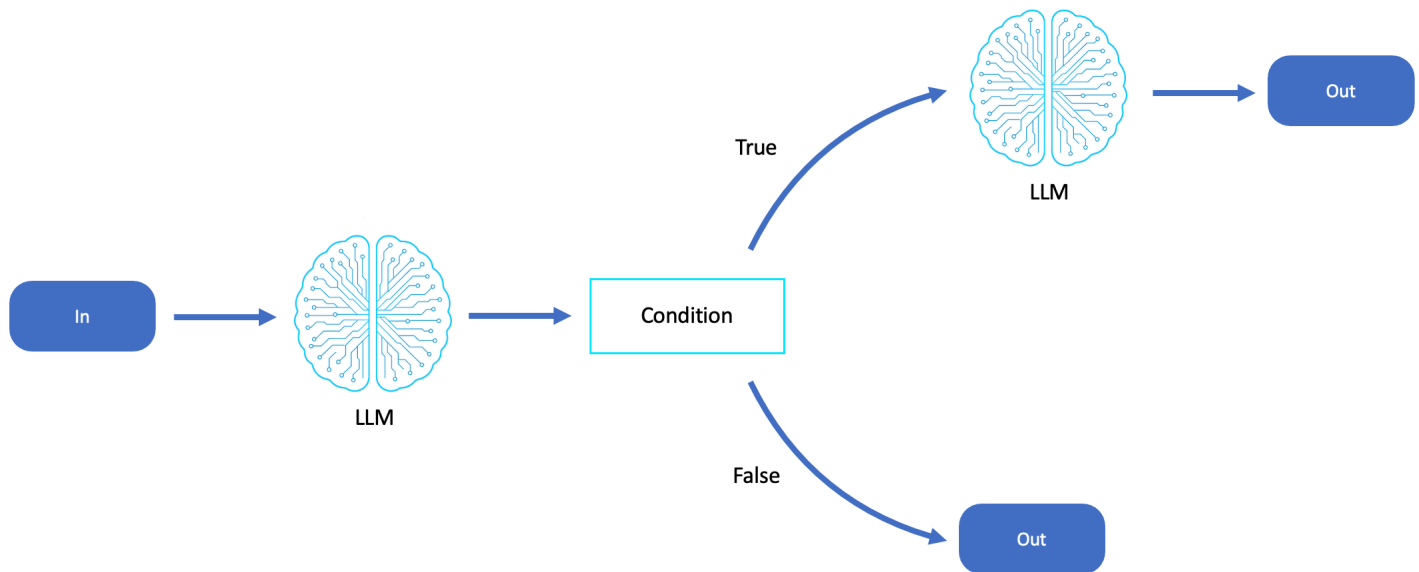
- メモリ – 構造化データベースまたはコンテキスト概要を使用して、永続的またはセッションベースの状態を推論ループに組み込む

これらの拡張は、LLM が時間の経過とともに、タスク間でどのように使用されるかを定義するワークフローで構成され、ステートレスエンジンから動的推論エージェントに変換されます。



プロンプト連鎖のワークフロー

プロンプト連鎖は、複雑なタスクを一連のステップに分解します。各ステップは、前のステップの出力を処理または構築する個別の LLM 呼び出しです。



プロンプト連鎖ワークフローは、タスクを論理的にシーケンシャル推論ステップに分割でき、中間出力が次のステージを通知するシナリオに適しています。ドキュメントレビュー、コード生成、ナレッジ抽出、コンテンツ絞り込みなど、構造化思考、プログレッシブトランスフォーメーション、レイヤード分析を必要とするワークフローに優れています。

説明

- タスクの複雑さが、1 回の LLM 呼び出しのコンテキストウィンドウまたは推論深度を超えています。
- 1 つのステップ (分析、要約、計画など) からの出力は、フォローアップの決定または生成フェーズの入力になります。
- 推論段階 (監査可能な中間結果など) 全体で透明性と制御が必要です。
- ステップ間で外部検証、フィルタリング、またはエンリッチメントロジックをプラグインする場合。
- これは、研究エージェント、編集アシスタント、計画システム、マルチステージ副操縦士など、パイプラインスタイルの推論ループで動作するエージェントに最適です。

機能

- LLM 呼び出しの線形チェーンまたは分岐チェーン
- 構造化された入力として渡されるか、フォローアッププロンプトに埋め込まれた中間結果

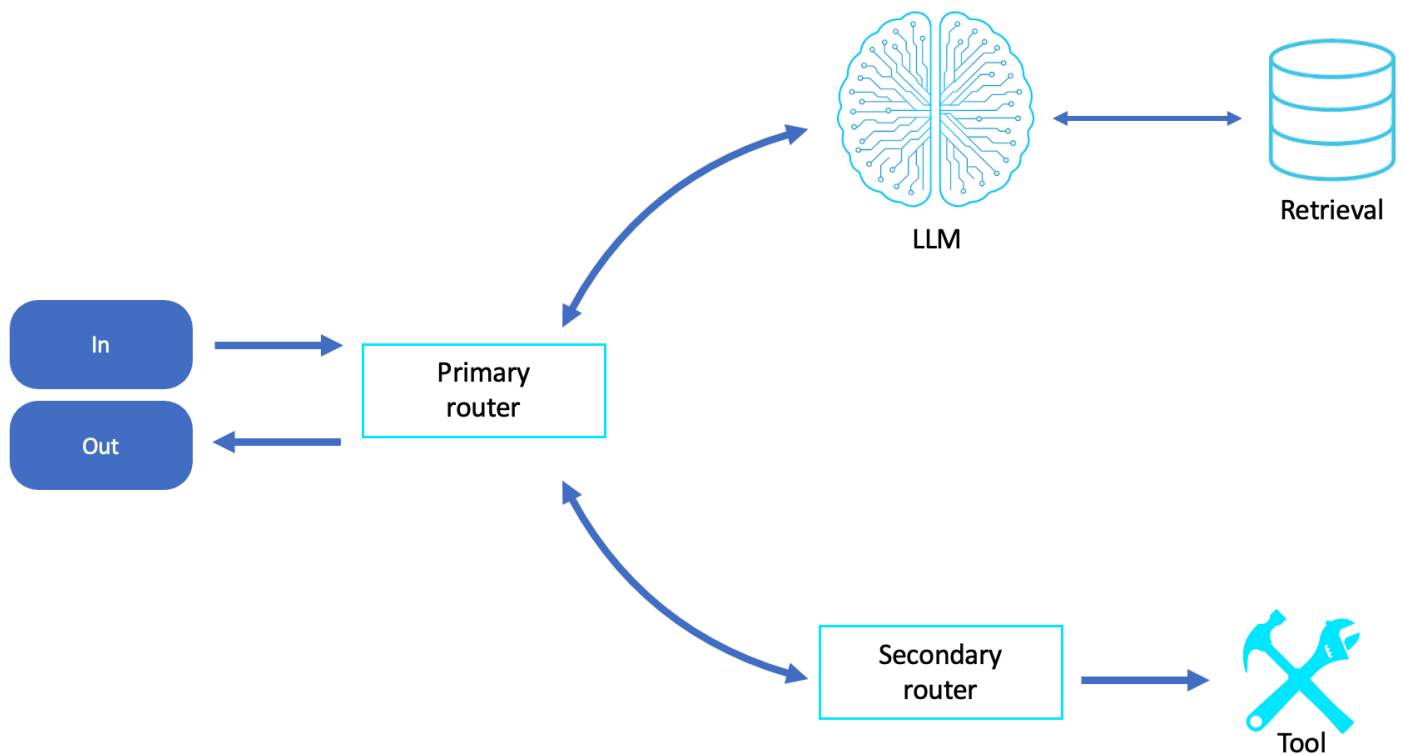
- AWS Step Functions AWS Lambda、またはエージェント固有のランナーとオーケストレーション可能

一般的なユースケース

- マルチステップ推論タスク (「批評書き換えの概要」など)
- レイヤー出力を合成する研究助手 (「検索抽出ファクトの回答の質問」など)
- コード生成パイプライン (「プラン書き込みコードテストコードの説明出力の生成」)

ルーティングのワークフロー

ルーティングパターンでは、分類子またはルーターエージェントは LLM を使用してクエリのインテントまたはカテゴリを解釈し、入力を特殊なダウンストリームタスクまたはエージェントにルーティングします。



ルーティングワークフローは、エージェントが入カインテント、タスクタイプ、またはドメインをすばやく分類し、リクエストを特殊なサブエージェント、ツール、またはワークフローに委任する必要があります。これは、一般アシスタント、エンタープライズ機能へのフロントド

ア、ドメインにまたがるユーザー向け AI インターフェイスなどの機能エージェントに特に役立ちます。

ルーティングは、次の場合に特に効果的です。

- さまざまなタスク (検索、要約、予約、計算など) にわたるリクエストのトリアージ。
- 入力は、より特殊なワークフローに入る前に前処理または正規化する必要があります。
- 入力タイプ (画像とテキスト、構造化クエリと非構造化クエリなど) が異なる場合は、カスタム処理が必要です。
- エージェントは会話型スイッチボードとして機能し、タスクを特殊なエージェントまたはマイクロサービスに委任します。
- このワークフローは、インテリジェントなディスパッチによってエージェントの動作の品質と効率の両方が決定される、ドメイン固有の副操縦士、カスタマーサポートボット、エンタープライズサービスルーター、マルチモーダルエージェントで一般的です。

機能

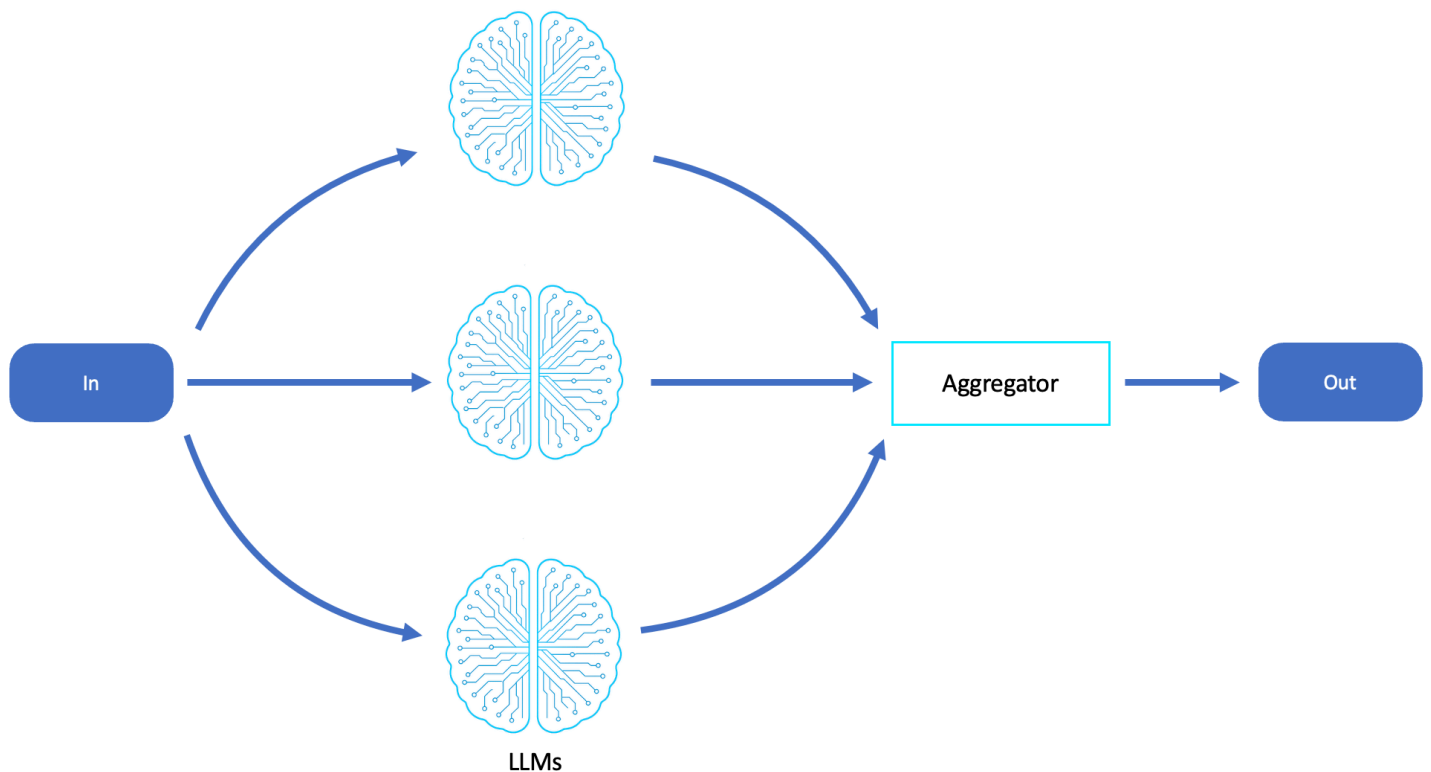
- ファーストパス LLM はディスパッチャーとして機能します
- ルートは個別のワークフローや他のエージェントパターンを呼び出すことができます
- 機能のモジュラー拡張をサポート

一般的なユースケース

- マルチドメインアシスタント (「これは法的、医療的、または財務的な質問ですか?」)
- LLM 推論で強化された決定木
- 動的ツールの選択 (検索とコード生成など)

並列化のワークフロー

このワークフローでは、タスクを独立したサブタスクに分割し、複数の LLM コールまたはエージェントで同時に処理できます。その後、出力はプログラムで集計され、結果に合成されます。



並列化ワークフローは、タスクを独立した非連続サブタスクに分割して同時に処理できる場合に使用します。これにより、効率、スループット、スケーラビリティが大幅に向上します。これは、エージェントが複数の入力にわたってコンテンツを分析または生成する必要がある、データ量の多い、バッチ指向、またはマルチパーспекティブな問題スペースで特に強力です。

並列化は、次の場合に特に効果的です。

- サブタスクは相互の中間結果に依存しないため、調整せずに並行して実行できます。
- タスクには、多くの項目にわたって同じ推論プロセスを繰り返すことが含まれます (複数のドキュメントを要約したり、オプションのリストを評価したりするなど)。
- 多様性、創造性、堅牢性を促進するために、複数の仮説や視点が並行して検討されます。
- LLM の同時実行により、高ボリュームまたは高頻度のリクエストのレイテンシーを減らす必要があります。
- このワークフローは、ドキュメント処理エージェント、調査または比較エンジン、バッチサマリ、マルチエージェントブレインストーミング、スケーラブルな分類またはラベル付けタスクで一般的に使用されます。特に、迅速で並行した推論がパフォーマンス上の利点である場合です。

機能

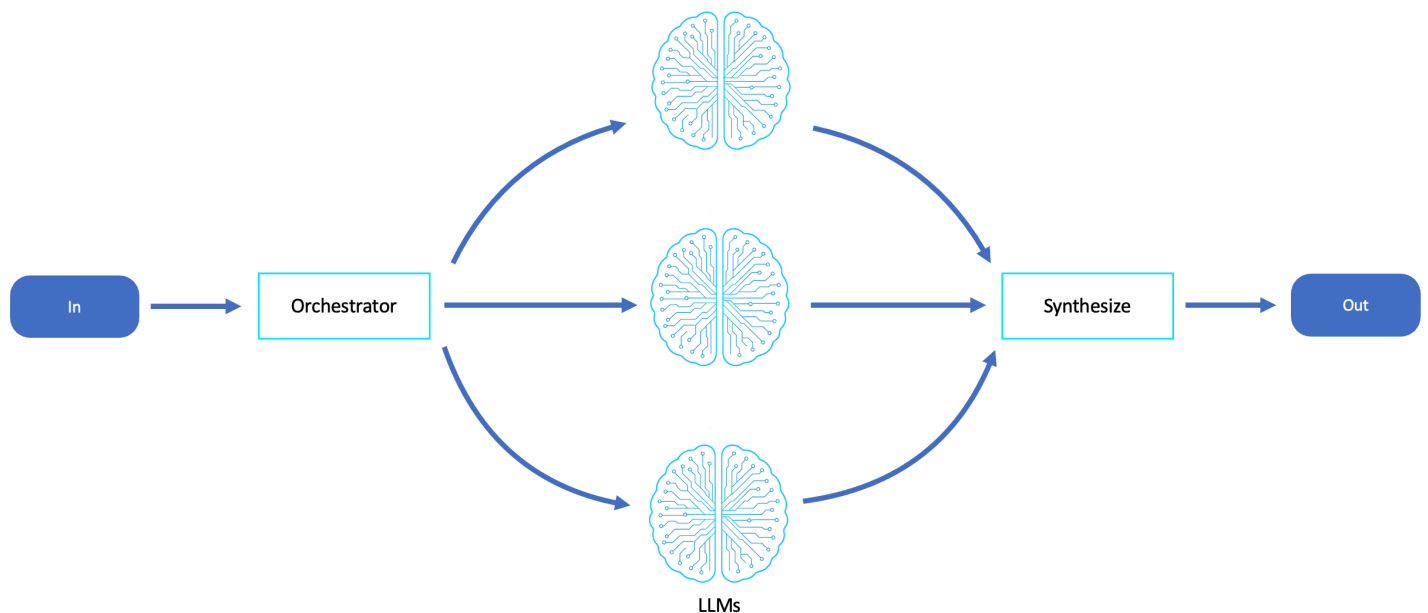
- LLM タスクの並列実行 (AWS Lambda、AWS Fargate、または AWS Step Functions マップ状態を使用)
- 合成段階で結果の整列、検証、または重複排除が必要です
- ステートレスエージェントループに最適

一般的なユースケース

- 複数のドキュメントまたはパースペクティブを並行して分析する
- 多様なドラフト、概要、または計画の生成
- バッチジョブ間のスループットの高速化

オーケストレーションのワークフロー

中央オーケストレーターエージェントは LLM を使用して、特定のロールまたはドメインの専門知識を持つ専門のワーカーエージェントまたはモデルにサブタスクを計画、分解、委任します。これは人間のチーム構造を反映し、複数のエージェントにわたる緊急の動作をサポートします。



オーケストレーションワークフローは、構造化された分解と特殊な実行を必要とする、複雑、階層的、または学際的なシナリオに最適です。これは、作業の分割を必要とするタスクに特に適してお

り、タスクのさまざまなサブコンポーネントは、異なる機能、知識、またはツールセットを持つエージェントによって最も適切に処理されます。

このワークフローは、次の場合に特に効果的です。

- タスクは、範囲、タイプ、または推論 (計画、研究、実装、テストなど) が異なるサブタスクに分割できます。
- LLM またはメタエージェントは、他のエージェントを調整し、進行状況をモニタリングし、結果を合成する必要があります。
- エージェントの責任をモジュール化して、スケーラビリティ、再利用、特殊なチューニングを実現したいと考えています。
- このシステムには、人間のチーム (プロジェクトマネージャー、開発者、レビュー担当者など) がコラボレーションでどのように動作するかを模倣した、ロールベースの動作が必要です。

オーケストレーションは、マルチターンプランニングエージェント、ソフトウェア開発副操縦士、エンタープライズプロセスエージェント、自律型プロジェクトエグゼキューターに最適です。これは、一元化されたタスクの内訳を必要とするが、実行ロジックが分散されたマルチエージェントシステムを実装する場合に特に便利です。これにより、エージェントレイヤー間で拡張性とより説明しやすい動作が可能になります。

機能

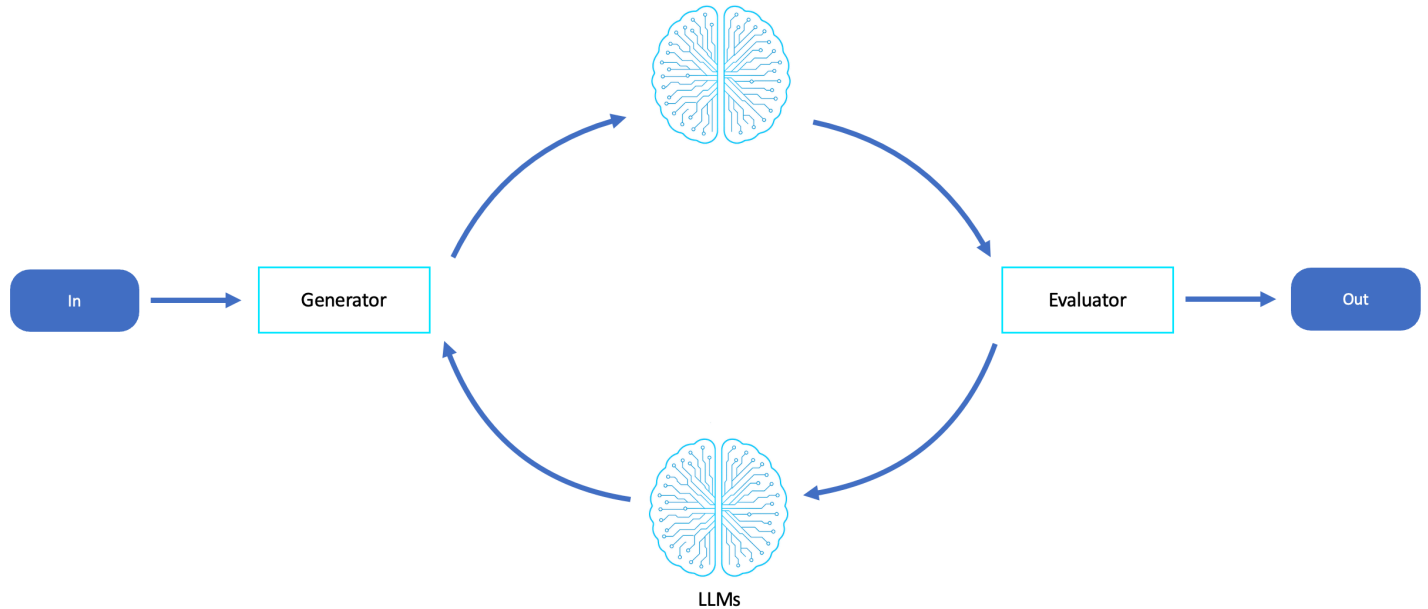
- オーケストレーターが目標メタ推論を実行する
- ワーカーエージェントには、ツールアクセス、メモリ、またはドメイン固有のプロンプトが含まれる場合があります
- 階層 (複数レベルのタスク委任) にすることができます

一般的なユースケース

- プロジェクトマネージャー、調整研究者、ライター、品質保証エージェント
- 計画、実行、テストを組み合わせたコーディング副操縦士
- ツールチェーンまたは API アクセスパターンを監督するエージェント

評価者とリフレクション/絞り込みループのワークフロー

このワークフローは、ある LLM が結果を生成し、別の LLM が結果を評価または批評するフィードバックループを提供します。これにより、自己リフレクション、最適化、反復的な改善が促進されます。



評価者ワークフローは、出力の品質、精度、調整が重要で、単一パス生成の信頼性が低い、または不十分なシナリオに最適です。このワークフローは、より高い正確性基準を満たすため、またはフィードバックに基づいて改善された代替案を検討するために、エージェントが出力を自己批評、反復、改良する必要がある場合に優れています。

このワークフローは、次の場合に特に効果的です。

- 出力には、主観的な品質メトリクス (スタイル、トーン、読みやすさなど) または目標基準 (正確性、安全性、パフォーマンスなど) が含まれます。
- エージェントは、トレードオフを通じて推論し、制約を評価し、目標に向けて最適化する必要があります。
- 特に規制対象ドメイン、顧客向けドメイン、またはクリエイティブドメインでは、組み込みの冗長性と品質保証が必要です。
- ヒューマン-in-the-loopレビューは高価または利用できないため、自律的な検証が必要です。

このワークフローは、コンテンツ生成、コード合成とレビュー、ポリシーの適用、アライメントチェック、指示調整、RAG 後処理に使用されます。また、継続的なフィードバックが時間の経過と

ともにより良い対応を形成し、信頼できる自律的な決定ループを構築するのに役立つ、自己改善エージェントにも役立ちます。

一般的なユースケース

- ブルーチームエージェントと比較したレッドチームエージェント
- コードまたは計画を生成、評価、および改訂するエージェント
- 品質保証、幻覚検出、スタイルの適用

機能

- さまざまなモデルを使用した分離された生成と評価をサポート (生成には Claude、評価には Mistral など)
- フィードバックは構造化され、改訂された出力を促すために使用されます。
- 複数の反復または収束のしきい値をサポート

結論

LLMs最新のソフトウェアエージェントのコグニティブコアを提供しますが、raw モデルの呼び出しでは、目的意識があり、堅牢で、制御可能なインテリジェンスを実現するには不十分です。出力生成から構造化された推論と目標に沿った動作に移行するには、モデルが入力进行处理し、コンテキストを管理し、アクションを調整する方法を定義する意図的なワークフローパターンに LLMs を埋め込む必要があります。

LLM ワークフローは、エージェントの認知モジュールを構築するための基盤を導入します。

- プロンプトの連鎖は、複雑な推論をモジュール式の監査可能なステップに分割します。
- ルーティングにより、インテリジェントなタスク分類とターゲットを絞った委任が可能になります。
- 並列化はスループットを加速し、多様な推論を促進します。
- エージェントオーケストレーションは、タスク分解とロールベースの実行を通じてマルチエージェントコラボレーションを構築します。
- 評価者 (リフレクトリファインループ) は、自己改善、品質管理、およびアライメントチェックを可能にします。

各ワークフローは、エージェントのニーズ、タスクの複雑さ、ユーザーの期待に適応できる構成可能なパターンを表します。これらのワークフローは相互に排他的ではありません。これらは、多くの場合、動的推論、マルチエージェント調整、エンタープライズグレードの信頼性をサポートするハイブリッドアーキテクチャに結合されるブロックを構築しています。

エージェントワークフローパターンに関する次の章に移行すると、これらの LLM ワークフローはより大きなシステム内の埋め込み構造として再び表示され、目標の委任、ツールオーケストレーション、決定ループ、ライフサイクルの自律性がサポートされます。これらの LLM ワークフローをマスターすることは、テキストを予測するだけでなく、理由、適応、意図的な行動を行うソフトウェアエージェントを設計するために不可欠です。

エージェントワークフローパターン

エージェントワークフローパターンは、モジュラーソフトウェアエージェントを構造化大規模言語モデル (LLM) ワークフローと統合し、自律的な推論とアクションを可能にします。従来のサーバーレスアーキテクチャとイベント駆動型アーキテクチャから着想を得たこれらのパターンは、コアロジックを静的コードから LLM 拡張エージェントに移行し、適応性とコンテキストに応じた意思決定を強化します。この進化により、従来のクラウドアーキテクチャは、スケーラビリティと応答性の基本的な原則を維持しながら、決定論的なシステムから動的な解釈とインテリジェントな拡張が可能なシステムに変換されます。

このセクションの内容

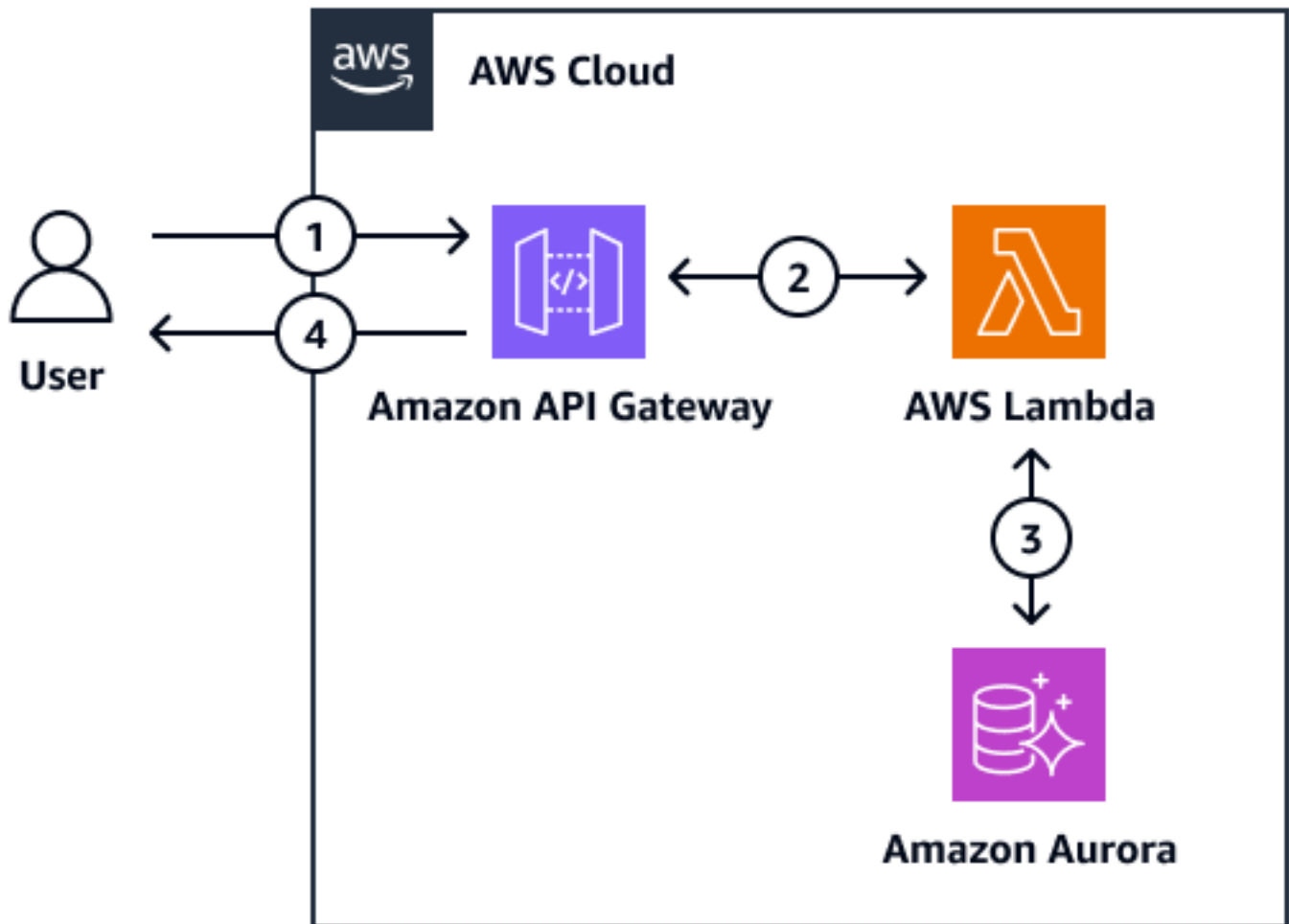
- [イベント駆動型システムから認識拡張型システムへ](#)
- [Saga パターンのプロンプト連鎖](#)
- [動的ディスパッチパターンのルーティング](#)
- [並列化パターンと散布図パターン](#)
- [Saga オーケストレーションパターン](#)
- [評価者の反射/絞り込みループパターン](#)
- [でのエージェントワークフローの設計 AWS](#)
- [結論](#)

イベント駆動型システムから認識拡張型システムへ

最新のクラウドアーキテクチャ、特にサーバーレスおよびイベント駆動型の原則に基づいて構築されたアーキテクチャは、従来、応答性が高くスケーラブルなシステムを作成するために、ルーティング、ファンアウト、エンリッチメントなどのパターンに依存してきました。エージェント AI システムは、LLM で強化された推論と認知の柔軟性を中心に再構築しながら、これらの基盤の上に構築されます。このアプローチにより、より高度な問題解決機能と自動化機能が可能になり、クラウド環境での複雑なタスクの処理方法に変革をもたらす可能性があります。

イベント駆動型アーキテクチャ

次の図は、一般的な分散システムを示しています。

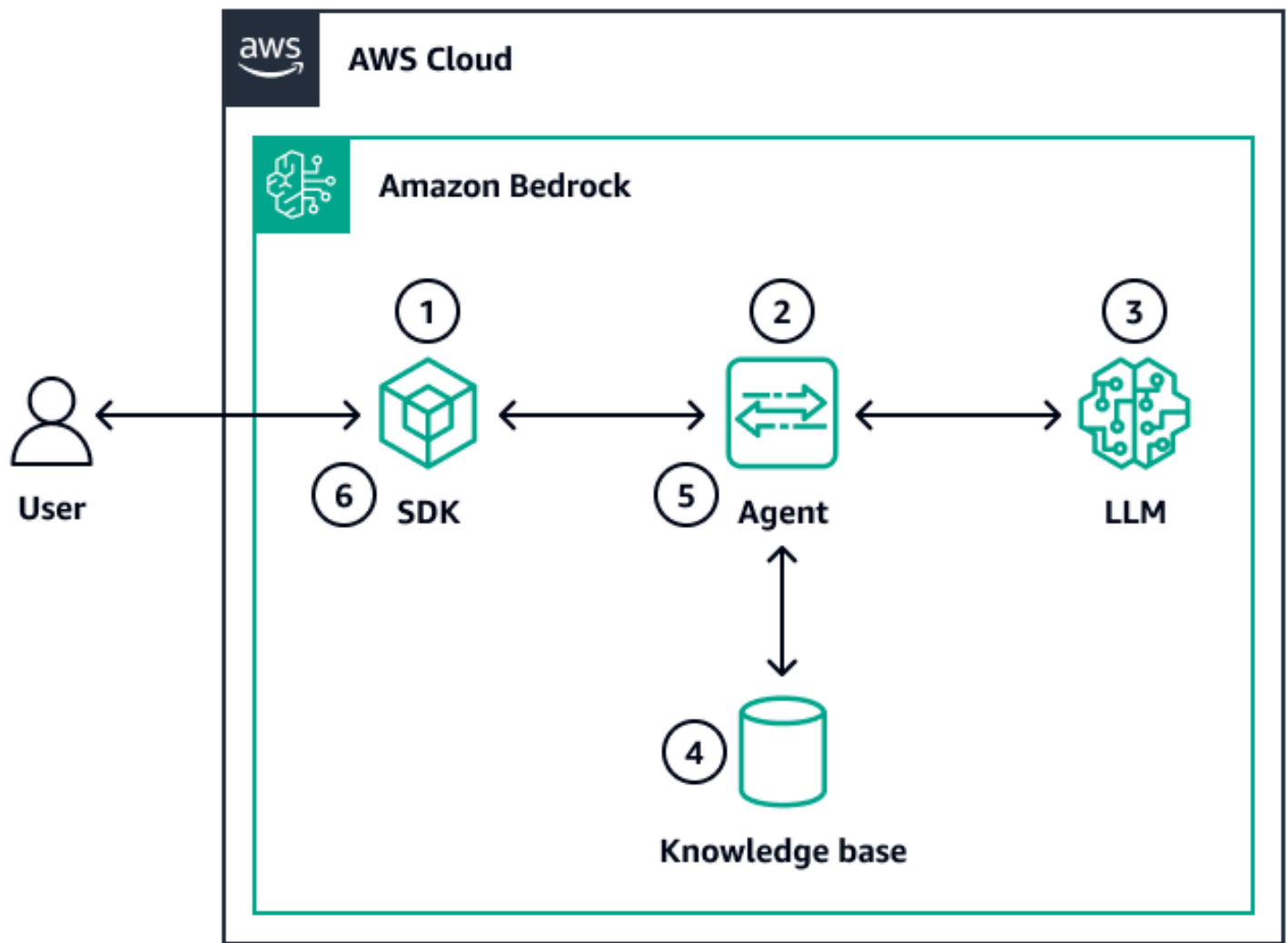


1. ユーザーは Amazon API Gateway にリクエストを送信します。
2. Amazon API Gateway は、リクエストを AWS Lambda 関数にルーティングします。
3. AWS Lambda Amazon Aurora データベースをクエリしてデータエンリッチメントを実行する
4. Amazon API Gateway は、エンリッチされたペイロードを呼び出し元に返します。

この構造は信頼性が高くスケーラブルですが、基本的には静的です。ビジネスルールとロジックパスは明示的にコーディングする必要があり、変化するコンテキストや不完全な情報への適応は制限されます。

認知拡張ワークフロー

エージェントアーキテクチャは、イベント駆動型システムに認知拡張を追加します。次の図は、同等のエージェントを示しています。



1. ユーザーは SDK または API コールを通じてクエリを送信します。
2. Amazon Bedrock エージェントはクエリを受け取ります。
3. エージェントは LLM を呼び出してクエリを解釈します
4. エージェントは、Amazon Bedrock ナレッジベースまたは他の外部データソースを検索してセマンティックエンリッチメントを実行します。
5. LLM は、コンテキスト豊富な目標に沿ったレスポンスを合成します。
6. システムは合成されたレスポンスをユーザーに返します。

このフローでは、LLM はロジックを使用し、インテントを理解し、関連するコンテキストを取得して組み合わせ、最適な応答方法を決定します。このパターンは、メッセージをさらにルーティングする前に外部データで拡張される従来のエンリッチメントパターンを反映しています。ただし、エー

エージェントシステムでは、このエンリッチメントは静的ルックアップではありません。代わりに、エンリッチメントは動的で、意味的にガイドされ、目的によって駆動されます。

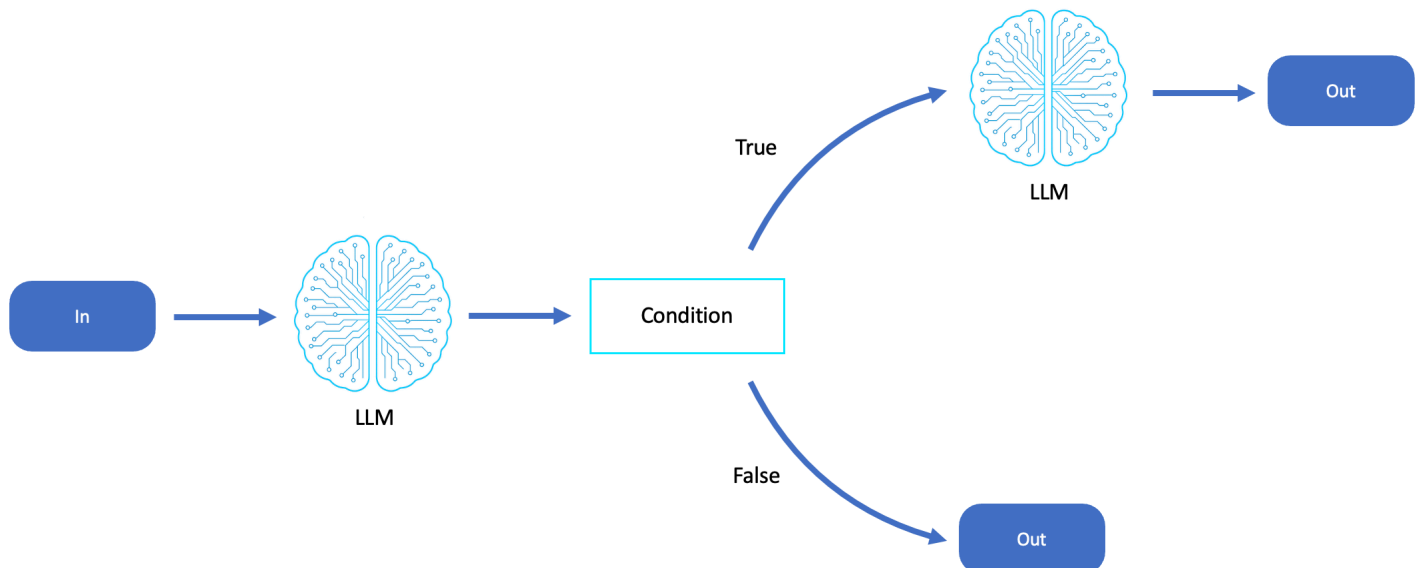
コアインサイト

各 LLM ワークフローは、従来のイベント駆動型アーキテクチャスタイルをミラーリングして進化させるエージェントワークフローパターンにマッピングできます。エージェントワークフローの基本的な構成要素は、LLM のコンテキストをデータ、ツール、メモリで拡張できることです。これにより、通知され、適応的で、ユーザーのインテントに沿った推論ループが作成されます。従来のシステムがルックアップデータでメッセージを強化する場合、エージェントシステムはソフトウェアがスクリプトのように動作したり、インテリジェントな共同作業者のように動作したりすることを可能にします。

Saga パターンのプロンプト連鎖

LLM プロンプトチェーンをイベント駆動型サガとして再考することで、ワークフローが分散され、回復可能になり、自律型エージェント間で意味的に調整される、新しい運用モデルが生まれます。各プロンプト/レスポンスステップは、アトミックタスクとして再フレームされ、イベントとして出力され、専用エージェントによって消費され、コンテキストメタデータで強化されます。

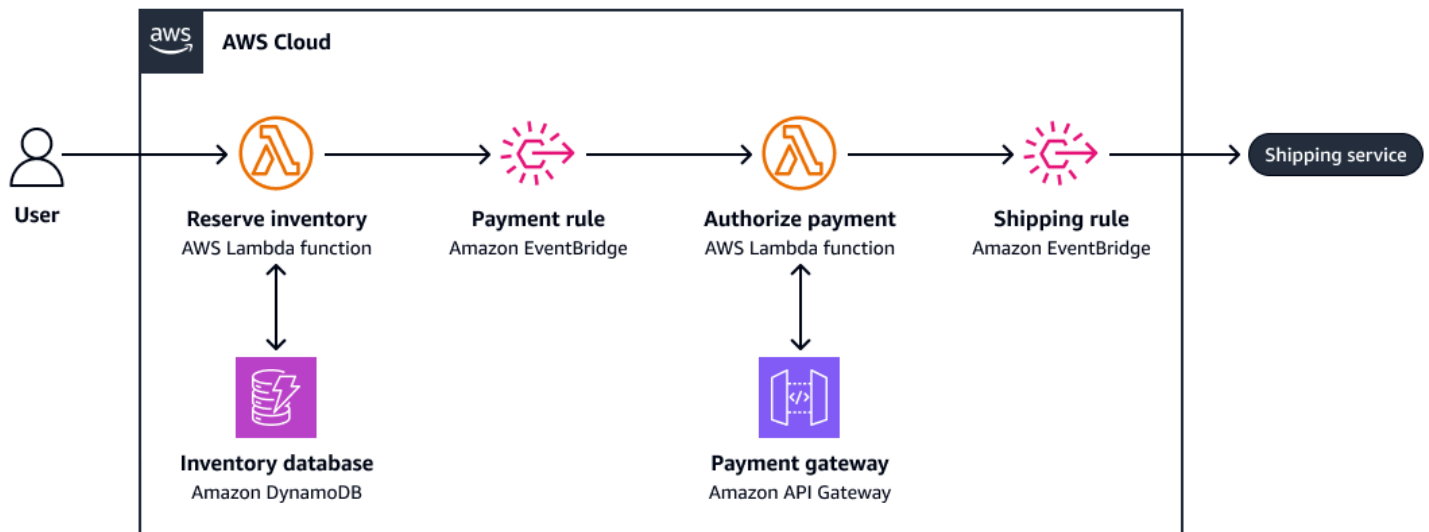
次の図は、LLM プロンプトの連鎖の例です。



Saga コレオグラフィ

Saga コレオグラフィパターンは、中央コーディネーターを持たない分散システムにおける実装アプローチです。代わりに、各サービスまたはコンポーネントは、次のワークフローアクションをトリガーするイベントを発行します。このパターンは、複数のサービス間でトランザクションを管理するために分散システムで広く使用されています。Saga では、システムは一連の調整されたローカルトランザクションを実行します。障害が発生した場合、システムは整合性を維持するために補償アクションをトリガーします。

次の図は、Saga コレオグラフィの例です。



1. インベントリを予約する
2. 支払いを承認する
3. 配送注文を作成する

ステップ 3 が失敗すると、システムは補償アクション (支払いのキャンセルやインベントリのリリースなど) を呼び出します。

このパターンは、サービスが疎結合され、部分的な障害がある場合でも、時間の経過とともに状態を一貫して解決する必要があるイベント駆動型アーキテクチャで特に役立ちます。

プロンプトの連鎖パターン

プロンプトの連鎖は、構造と目的の両方で saga パターンに似ています。コンテキストを保持し、ロールバックとリビジョンを許可しながら、順次構築する一連の推論ステップを実行します。

エージェントコレオグラフィ

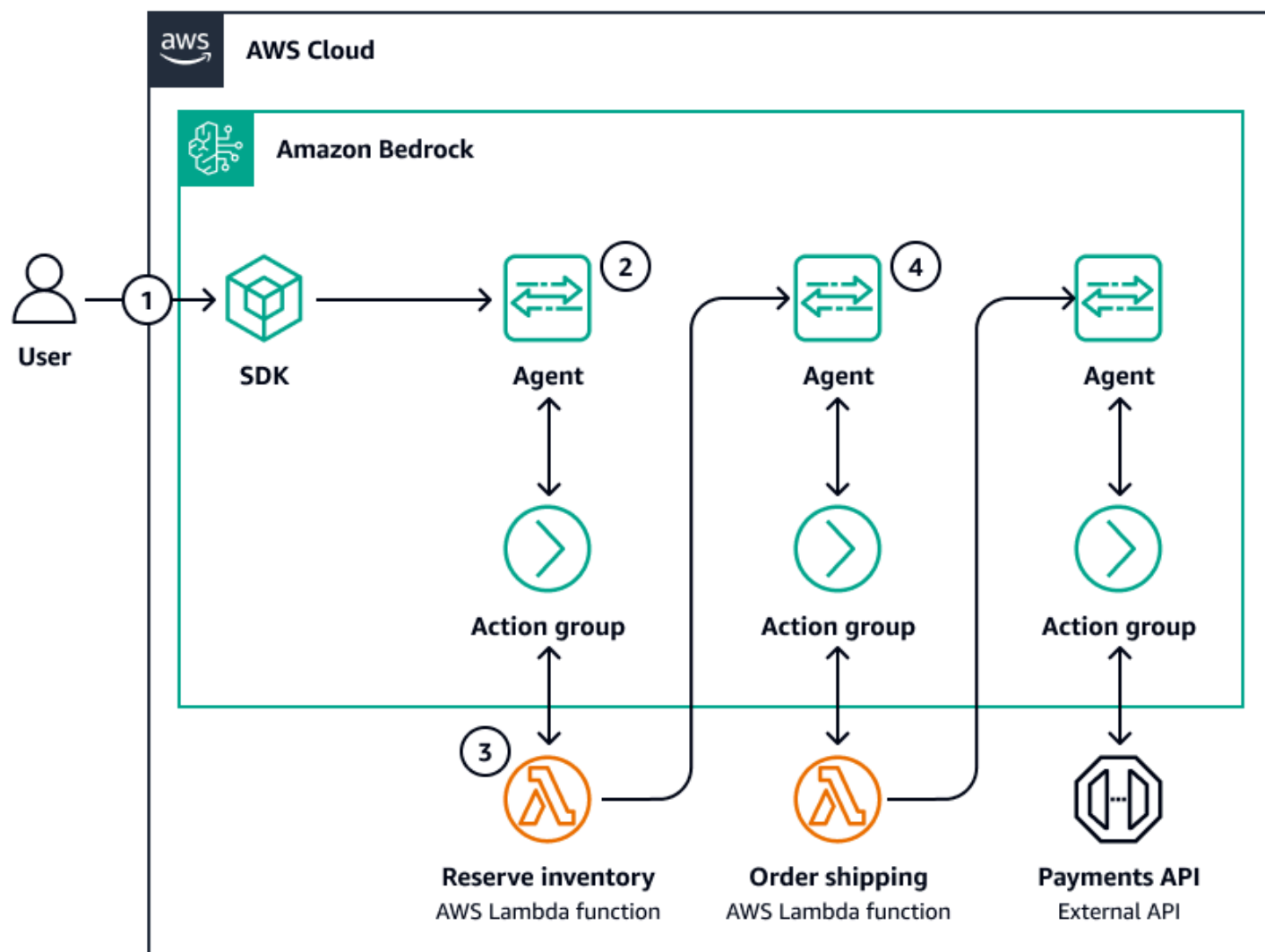
1. LLM は複雑なユーザークエリを解釈し、仮説を生成します
2. LLM がタスクを解決するための計画を策定する
3. LLM がサブタスクを実行する (ツール呼び出しの使用や知識の取得など)
4. LLM は、結果が不十分であると判断した場合、出力を絞り込むか、前のステップを再検討します。

中間結果に欠陥がある場合、システムは次のいずれかを実行できます。

- 別のアプローチを使用してステップを再試行する
- 前のプロンプトに戻って再計画する
- 評価者ループ (評価者とオプティマイザのパターンなど) を使用して障害を検出して修正する

Saga パターンと同様に、プロンプトの連鎖により部分的な進行とロールバックのメカニズムが可能になります。これは、データベーストランザクションを補償するのではなく、反復的な絞り込みと LLM 指向の修正によって行われます。

次の図は、エージェントのコレオグラフィの例です。



1. ユーザーは SDK を介してクエリを送信します。
2. Amazon Bedrock エージェントは、次のように推論を調整します。
 - 解釈 (LLM)
 - 計画 (LLM)
 - ツールまたはナレッジベースによる実行
 - レスポンスの構築
3. ツールが失敗するか、不十分なデータを返す場合、エージェントはタスクを動的に再計画または再フレーズできます。
4. メモリ (短期ベクトルストアなど) はステップ間で状態を保持できます

重要事項

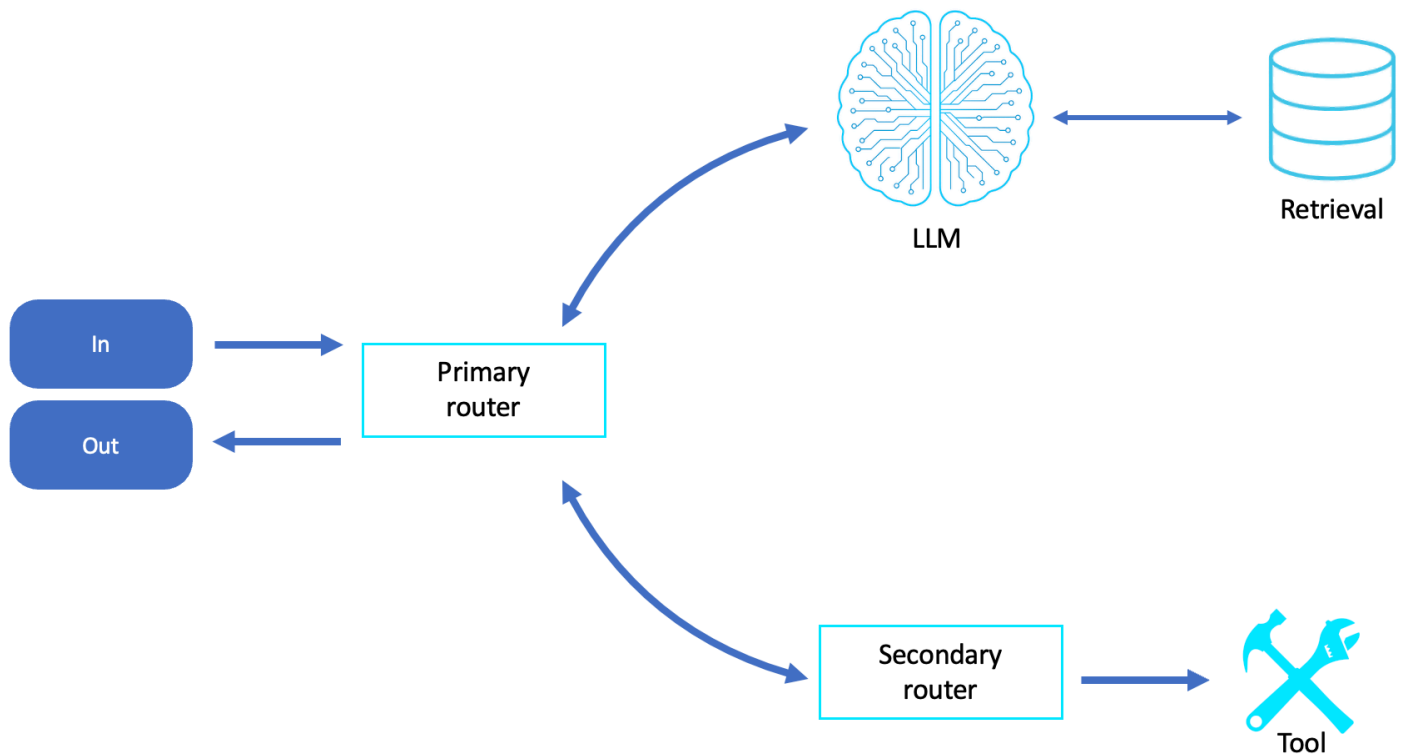
Saga パターンが補償ロジックを使用して分散型サービス呼び出しを管理する場合、プロンプトチェーンはリフレクティブシーケンスと適応型再計画を使用して推論タスクを管理します。どちらのシステムも、段階的な進行、分散された決定ポイント、障害復旧を可能にし、厳格なロールバックではなく、情報に基づいた推論を通じてこれらをすべて実行します。

プロンプト連鎖では、トランザクションの推論が導入されます。これは、Sagas と認識的に同等です。つまり、より広範な目標指向の対話の一環として、各「思考」が再評価、改訂、または中止されます。

動的ディスパッチパターンのルーティング

最新のエージェントシステムでは、タスクがドキュメント解析から自律的なソフトウェア生成まで多岐にわたるため、最も能力の高い大規模言語モデル (LLM) またはエージェントにリクエストを動的にルーティングする能力が重要になります。多くの場合、オーケストレーションスクリプトまたは API レイヤーに埋め込まれている静的ルーティングロジックには、リアルタイム、マルチモデル、マルチ機能環境に必要な適応性がありません。これに対処するために、LLM ルーティングワークフローを動的ディスパッチパターンを活用するイベント駆動型アーキテクチャに変換し、LLM 呼び出しをインテリジェントにルーティングされたコンテキスト対応イベントに変換できます。

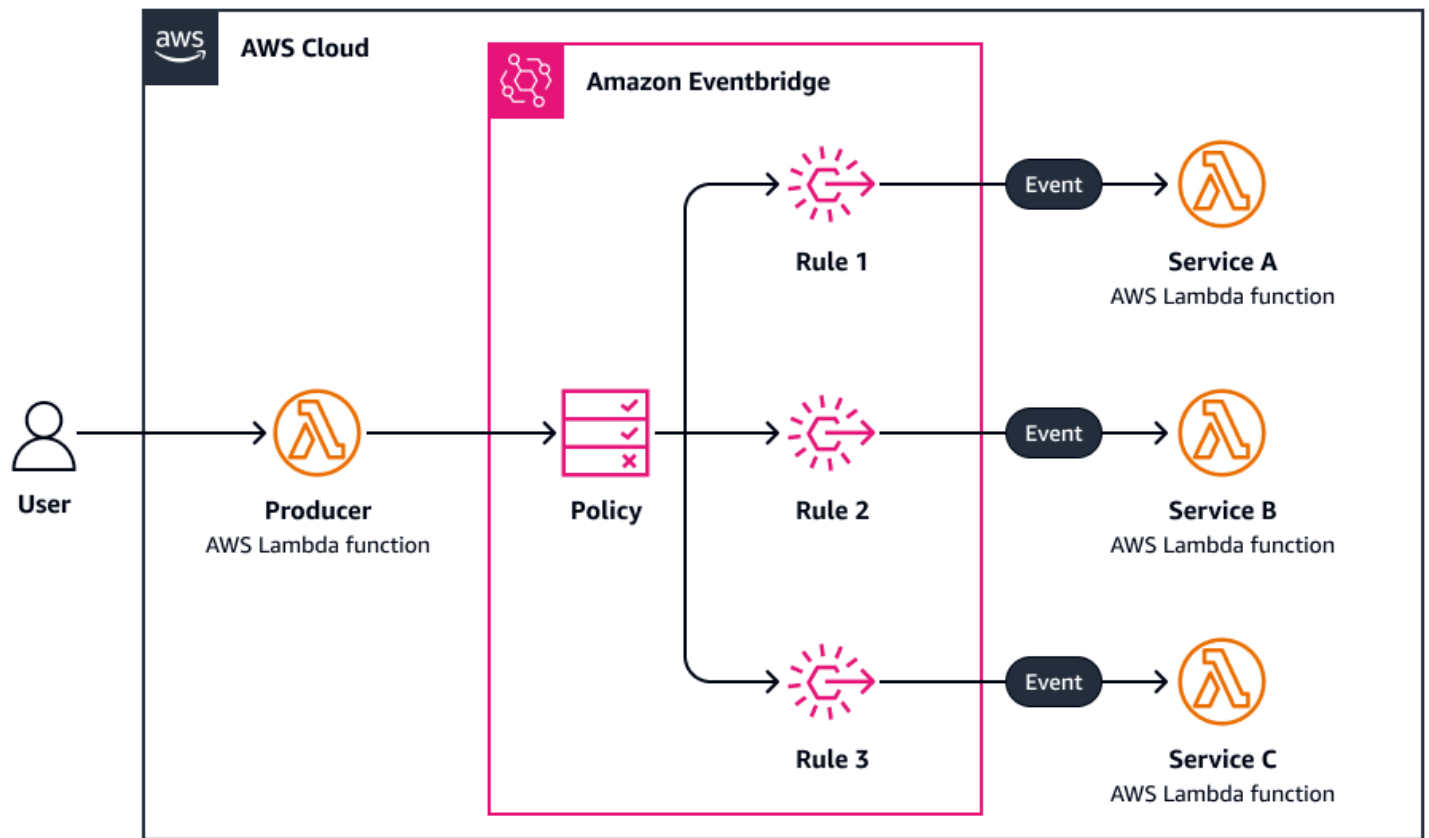
次の図は、LLM ルーティングの例です。



動的ディスパッチ

従来の分散システムでは、動的ディスパッチパターンは、イベントタイプ、ソース、ペイロードなどの受信イベント属性に基づいて、実行時に特定のサービスを選択して呼び出します。これは通常、Amazon EventBridge を使用して実装されます。Amazon EventBridge では、受信イベントを評価して適切なターゲット (AWS Lambda 関数、AWS Step Functions、Amazon Elastic Container Service タスクなど) にルーティングできます。

次の図は、動的ディスパッチの例です。



1. アプリケーションはイベントを発行します (例: `{"type": "orderCreated", "priority": "high"}`)。
2. Amazon EventBridge は、ルーティングルールに照らしてイベントを評価します。
3. イベントの属性に基づいて、システムは次のように動的にディスパッチします。
 - HighPriorityOrderProcessor (サービス A)
 - StandardOrderProcessor (サービス B)
 - UpdateOrderProcessor (サービス C)

このパターンは、疎結合、ドメインベースの特殊化、ランタイムの拡張性をサポートしています。これにより、システムは変化する要件やイベントセマンティクスにインテリジェントに対応できます。

LLM ベースのルーティング

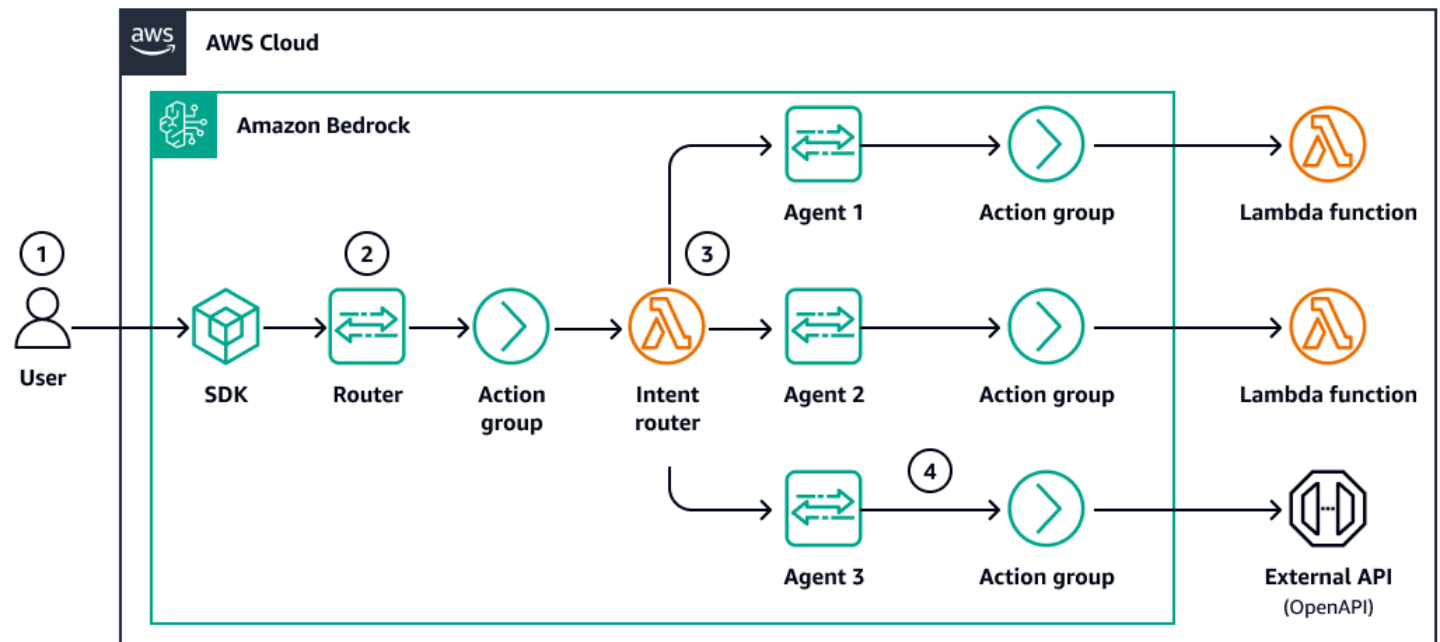
エージェントシステムでは、ルーティングは動的タスク委任も実行しますが、Amazon EventBridge ルールやメタデータフィルターの代わりに、LLM は自然言語を通じてユーザーのインテントを分類して解釈します。その結果、柔軟でセマンティック、適応的なディスパッチ形式になります。

エージェントルーター

このアーキテクチャにより、事前定義されたスキーマやイベントタイプなしで豊富なインテントベースのディスパッチが可能になります。これは、非構造化入力や複雑なクエリに最適です。

1. ユーザーがリクエストを送信します「契約条件の確認をお手伝いいただけますか？」
2. LLM はこれを法的ドキュメントタスクとして解釈します。
3. エージェントは、タスクを次のいずれかにルーティングします。
 - 契約レビュープロンプトテンプレート
 - 法的推論サブエージェント
 - ドキュメント解析ツール

次の図は、エージェントルーターの例です。



1. ユーザーは SDK を介して自然言語リクエストを送信します。
2. Amazon Bedrock エージェントは、LLM を使用してタスク (法律、技術、スケジューリングなど) を分類します。
3. エージェントは、アクショングループを介してタスクを動的にルーティングして、必要なエージェントを呼び出します。
 - ドメイン固有のエージェント
 - 特殊なツールチェーン

- カスタムプロンプト設定

4. 選択したハンドラーはタスクを処理し、カスタマイズされたレスポンスを返します。

重要事項

従来の動的ディスパッチでは、構造化イベント属性に基づくルーティングに Amazon EventBridge ルールが使用されますが、エージェントルーティングでは LLMs を意味論的に分類およびルーティングします。これにより、以下を有効にすることでシステムの柔軟性が向上します。

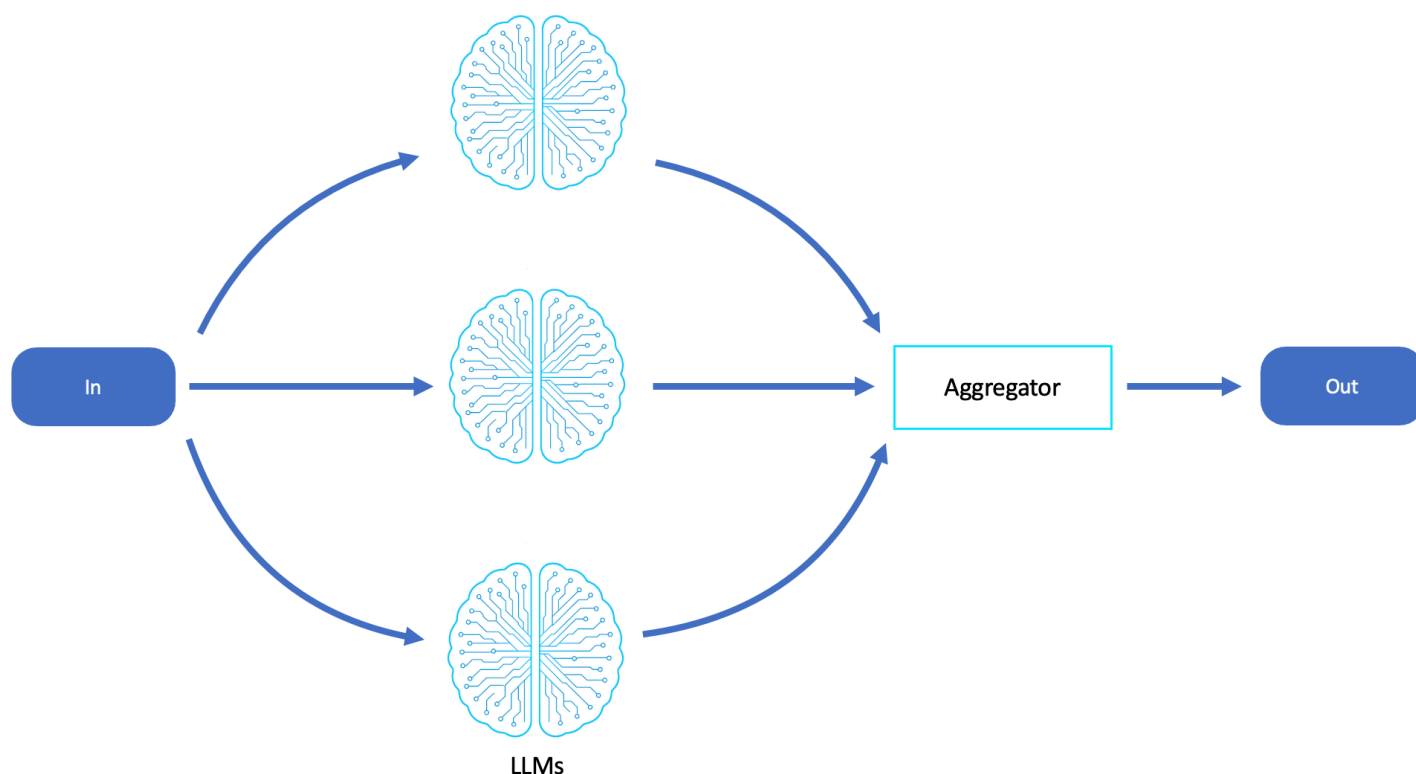
- より広範な入力の理解
- インテリジェントなフォールバックとツールの選択
- 新しいエージェントロールまたはプロンプトスタイルによる自然な拡張性

エージェントルーティングは、リジッドルールを動的コグニティブディスパッチに置き換えます。これにより、システムはコードではなく言語で進化できます。

並列化パターンと散布図パターン

大きなドキュメントの要約、複数のソリューションパスの評価、多様な視点の比較など、多くの高度な推論および生成タスクは、プロンプトの並列実行からメリットを得ます。スケーラビリティ、応答性、耐障害性が必要な場合、従来のシーケンシャルワークフローは短くなります。これを克服するために、LLM ベースの並列化は、タスクが自律エージェントに動的にファンアウトされ、結果がインテリジェントに合成されるイベント駆動型の散布図パターンを使用して再考できます。

次の図は、LLM 並列化ワークフローの例です。



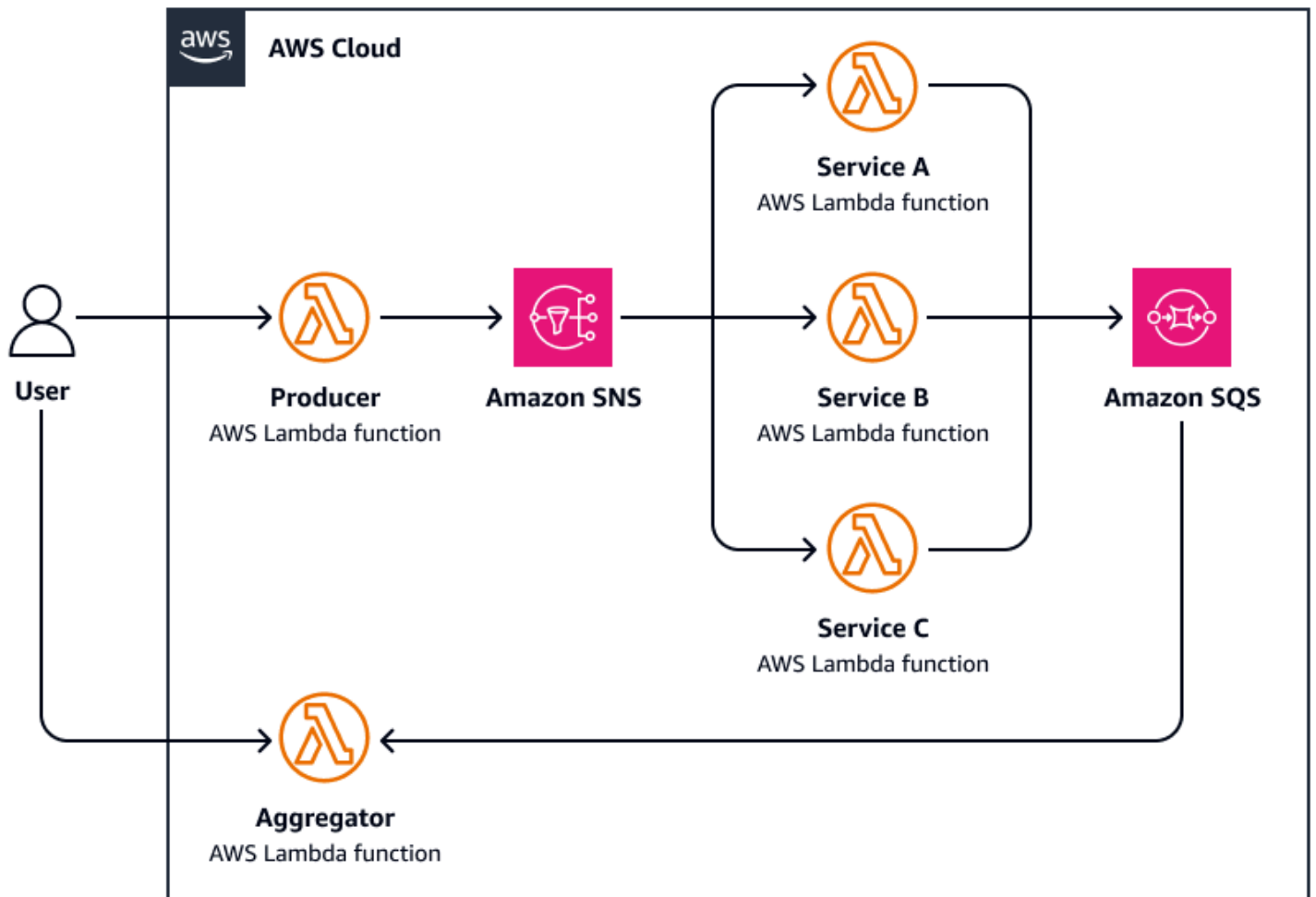
分散-収集

分散システムでは、分散図パターンはタスクを複数のサービスまたは処理ユニットに並行して送信し、応答を待機してから、結果を統合出力に集約します。ファンアウトとは異なり、scatter-gather はレスポンスを想定し、通常は結果を結合、比較、選択するロジックを適用するため、調整されます。

並列化と分散収集の一般的な実装は次のとおりです。

- AWS Step Functions 並列タスク実行の状態をマッピングする
- AWS Lambda 同時実行を使用して、複数の呼び出し関数の結果を調整する
- 関連 IDs と集約ワークフローを備えた Amazon EventBridge
- Amazon Simple Storage Service (Amazon S3)、Amazon DynamoDB、またはキューを使用してファンアウトを管理し、結果を収集するカスタムコントローラーパターン

次の図は、分散図の例です。



1. ユーザーは、Amazon Simple Notification Service (Amazon SNS) トピックに並列メッセージを発行してタスクを分散する中央コーディネーター関数にリクエストを送信します。
2. 各メッセージにはタスクメタデータが含まれ、特殊なワーカーにルーティングされます AWS Lambda。
3. 各ワーカーは、割り当てられたサブタスクを AWS Lambda 個別に処理します (外部 API のクエリ、ドキュメントの処理、データの分析など)。
4. 結果は、Amazon Simple Queue Service (Amazon SQS) などの一般的なストレージレイヤーに書き込まれます。
5. アグリゲータ関数は、すべてのレスポンスが完了するまで待機し、以下を実行します。
 - 結果を収集して集計します (概要をマージする、最適な一致を選択するなど)
 - 最終レスポンスを送信するか、ダウンストリームワークフローをトリガーします

散布的収集パターンの一般的なユースケースは次のとおりです。

- フェデレーティッド検索
- 価格比較エンジン
- 集計データ分析
- マルチモデル推論

LLM ベースの並列化 (散布図認識)

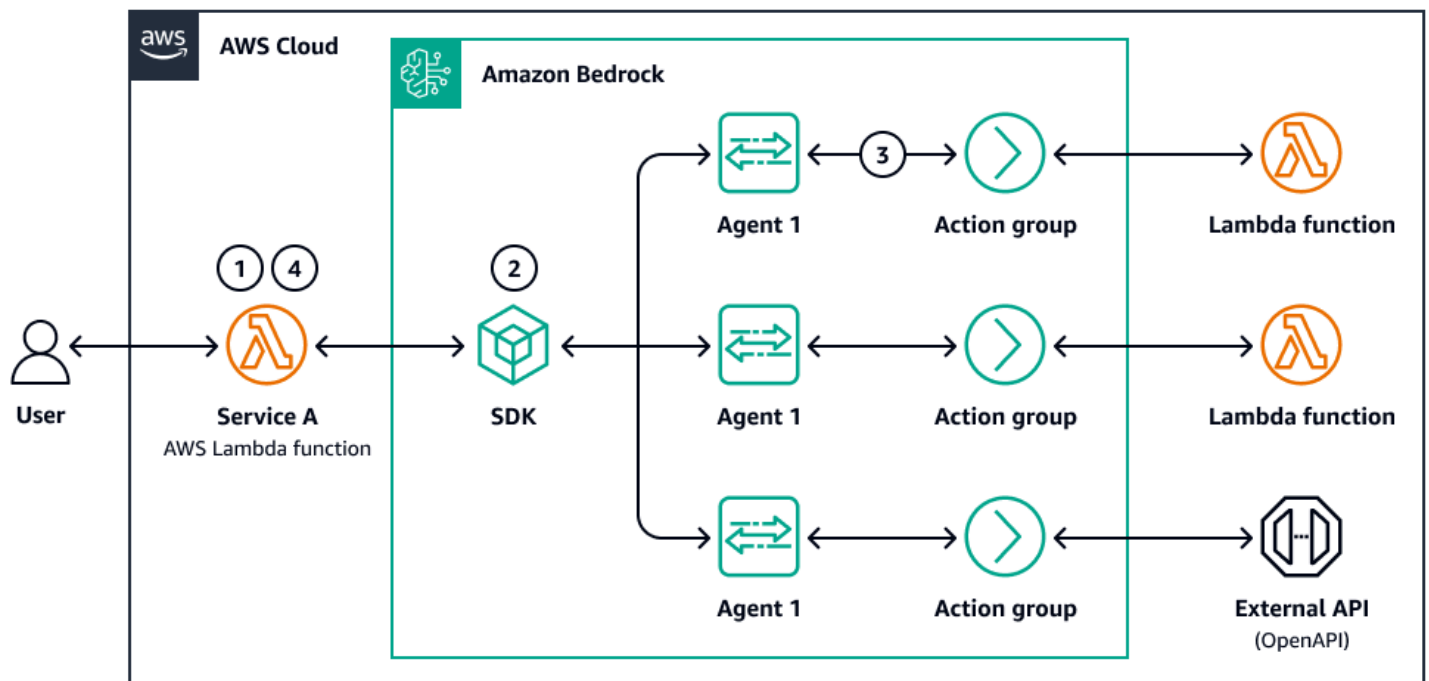
エージェントシステムでは、並列化は、複数の LLM コールまたはエージェントにサブタスクを分散することで散布図を密接にミラーリングし、それぞれが問題の一部を個別に推論します。返される結果は、多くの場合別の LLM またはコントローラーエージェントである集約プロセスによって収集および合成されます。

エージェント並列化

1. エージェントは、「これら 10 件のレポート全体でインサイトを要約する」リクエストを送信します。
2. レポートは 10 個の並列 LLM 要約タスクに分散されます。
3. すべての概要を返すと、エージェントは以下を実行します。
 - 概要を統一された説明に集約します
 - テーマまたは矛盾を特定する
 - 合成された出力をユーザーに送信します

このエージェントワークフローにより、スケーラブル、モジュール式、適応型の並列推論が可能になります。これは、高い認知スループットを必要とするユースケースに最適です。

次の図は、エージェントの並列化の例です。



1. ユーザーがマルチパートクエリまたはドキュメントセットを送信します。
2. コントローラー AWS Lambda またはステップ関数は、サブタスクを分散します。各タスクは、独自のプロンプトを使用して Amazon Bedrock LLM コールまたはサブエージェントを呼び出します。
3. 呼び出しとサブタスクが完了すると、結果が保存され (Amazon S3 やメモリストアなど)、集計ステップが出力をマージ、比較、またはフィルタリングします。
4. システムは、ユーザーまたはダウンストリームエージェントに最終レスポンスを返します。

このシステムには、トレーサビリティ、耐障害性、オプションの結果の重み付けまたは選択ロジックを備えた分散推論ループがあります。

重要事項

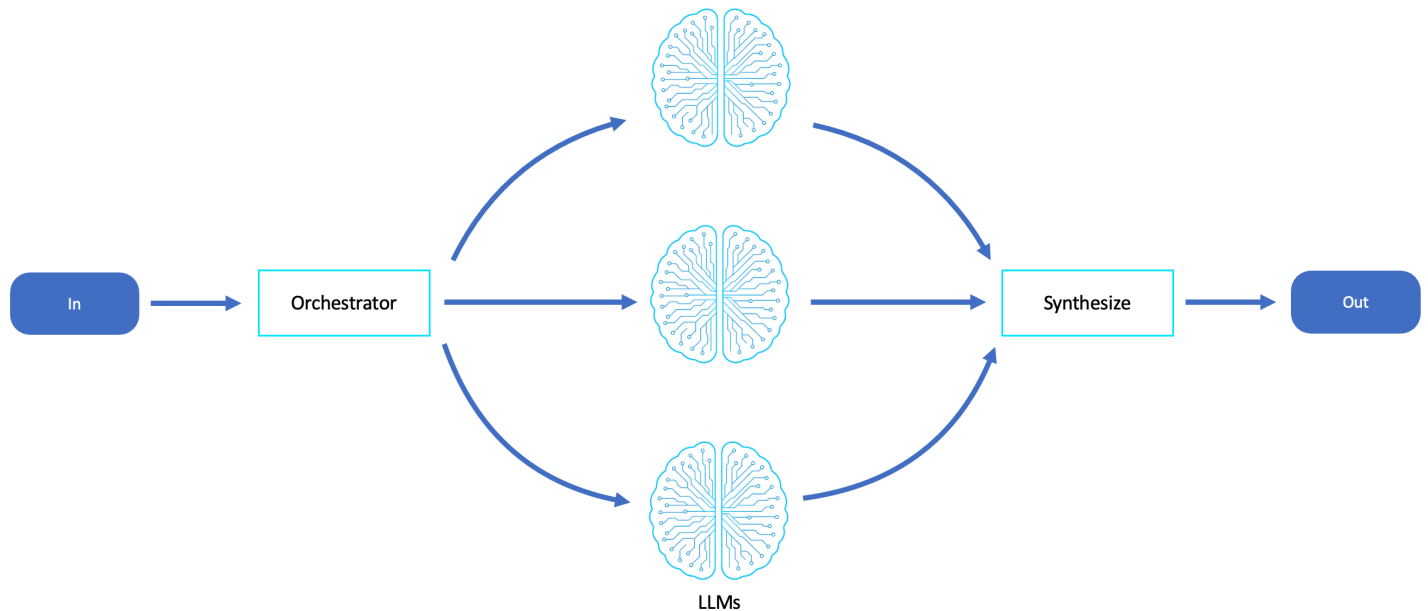
エージェント並列化は、散乱集合パターンを使用して LLM タスクを分散し、並列処理とインテリジェントな結果合成を可能にします。

Saga オーケストレーションパターン

LLMs によって駆動されるワークフローがますます複雑になり、プロンプトチェーン、データ処理ステップ、ツール呼び出し、エージェントのコラボレーションにまたがるにつれて、インテリジェント

なオーケストレーションの必要性が不可欠です。これらのワークフローは、緊密に結合されたスクリプトや静的な事前定義された実行フローに依存するのではなく、イベント駆動型のオーケストレーションパターンとして実装できるため、LLM ベースのシステムは自律型エージェント間でマルチステップタスクを動的に調整、モニタリング、適応できます。

次の図は、オーケストレーターの例です。



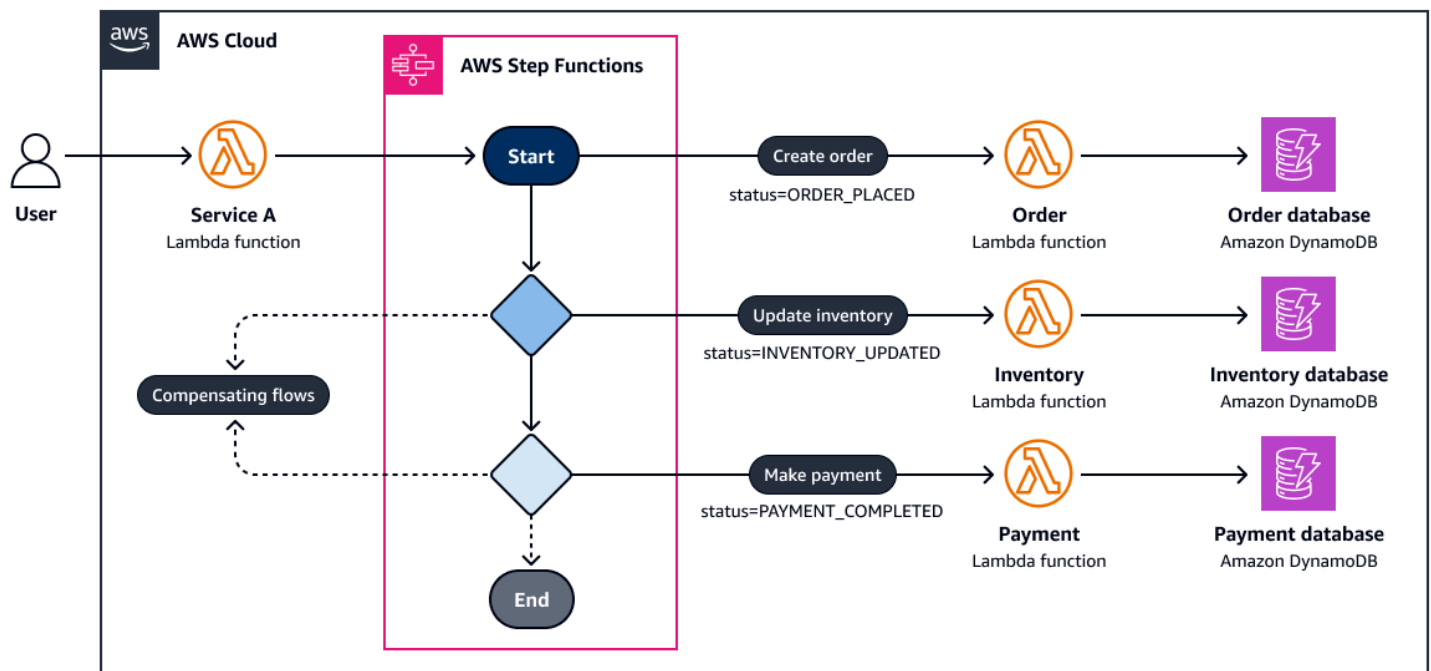
イベントオーケストレーション

従来の分散システムでは、イベントオーケストレーションとは、中央コーディネーターが複数のサービスまたはタスクにわたって制御フローを明示的に指示することで、複雑なワークフローを管理するパターンを指します。イベントコレオグラフィ (各サービスが独立して反応する) とは異なり、オーケストレーションはプロセス全体の一元化されたロジック、可視性、制御を提供します。

これは通常、次のツールを使用して実装されます。

- AWS Step Functions – ステートフルワークフローを定義して実行する
- AWS Lambda – オーケストレーションされたフロー内で個別のタスクを実行する
- Amazon SQS または Amazon EventBridge – 非同期ステップまたはレスポンスをトリガーします

次の図は、Saga オーケストレーションの例です。



AWS Step Functions ワークフローは、顧客注文プロセスを管理します。

1. 注文を作成する (AWS Lambda)
2. インベントリの更新 (AWS Lambda)
3. 支払いを行う (AWS Lambda)

オーケストレーターは、再試行、並列ブランチ、タイムアウト、失敗を管理することで、各ステップを調整します。

ロールベースのエージェントシステム (オーケストレーター)

エージェントシステムでは、オーケストレーターパターンはイベントオーケストレーションをミラーリングしますが、それぞれが定義されたロールまたは専門分野を持つ複数の推論エージェントにロジックを分散します。中央オーケストレーターエージェントは、タスク全体を解釈し、サブタスクに分解して、ワーカーエージェントに委任します。ワーカーエージェントはそれぞれ、特定のドメイン (研究、コーディング、要約、レビューなど) 用に最適化されています。

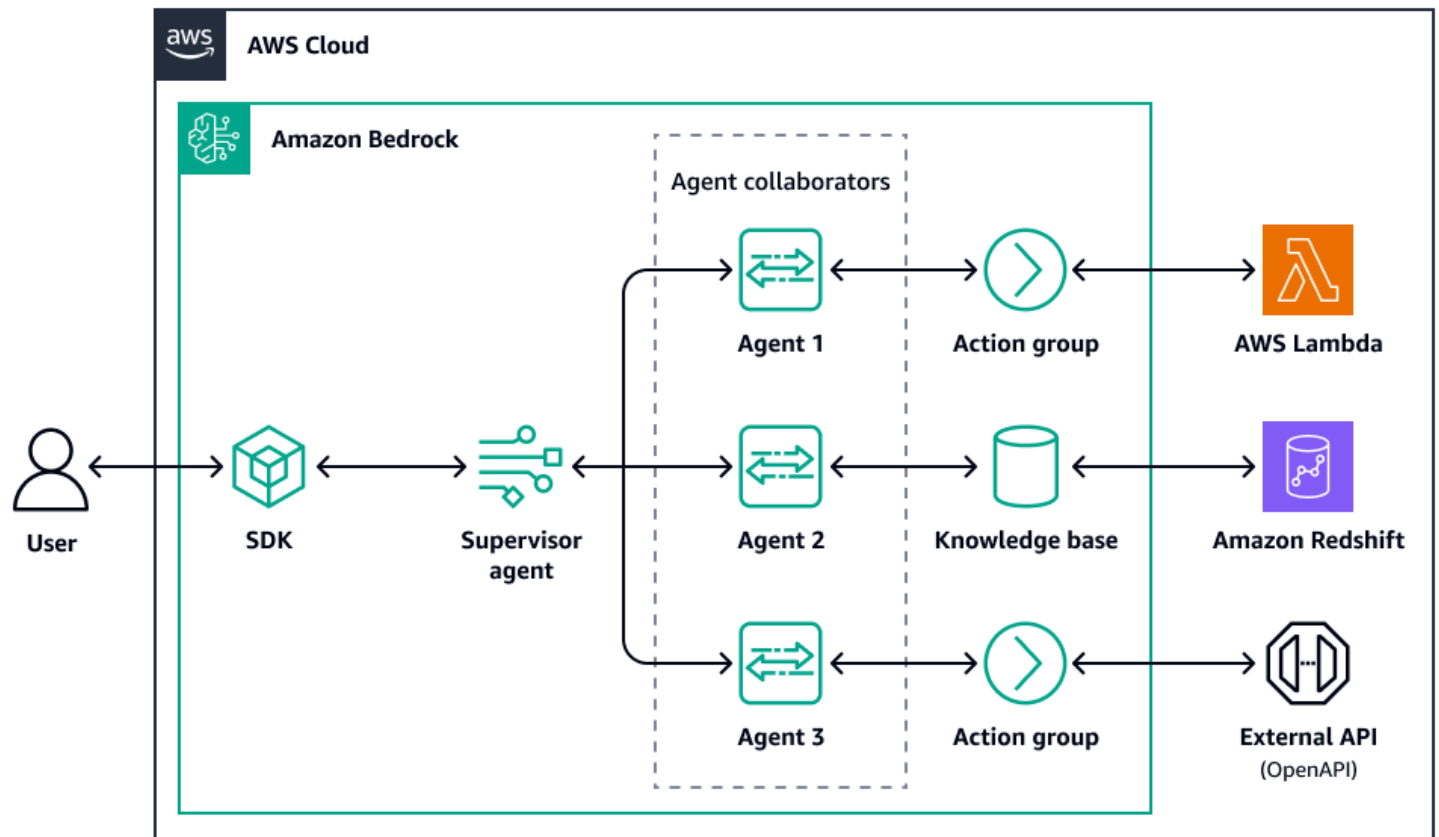
スーパーバイザー

1. ユーザーは「プロジェクト概要を作成し、上位 5 つの競合相手を要約する」というクエリを送信します。
2. オーケストレーターエージェントは以下を実行します。

- 競合データを見つけるために調査エージェントを割り当てます
- 未加工の検出結果を要約エージェントに送信します
- ショートライターエージェントに結果を渡す
- ユーザーの最終出力をコンパイルします

各エージェントは独立して動作しますが、オーケストレーターはタスクを調整します。これは、ワークフロータスクを処理する Lambda 関数のようなものです。

次の図は、スーパーバイザーの例です。



1. ユーザーは Amazon Bedrock スーパーバイザーエージェントにタスクを送信します。
2. スーパーバイザーエージェントは、エージェントコラボレーターごとにリクエストをサブタスクに解析します。
3. 各サブタスクは、ロール固有のプロンプトまたはツールチェーンを持つ共同作業エージェントに割り当てられます。
4. ワーカーエージェントAPIs またはツールを呼び出します。
5. 各ワーカーエージェントは、構造化形式で出力を返します。

6. すべてのワーカーが結果を返すと、スーパーバイザーは評価、合成、最終レスポンスを返します。

この構造により、複雑なマルチステップエージェントワークフロー全体でモジュール性、適応性、インタロスペクションが可能になります。

重要事項

イベントオーケストレーションが集中制御 (例: AWS Step Functions) を使用してサービス実行を指示する場合、ロールベースのエージェントシステムは LLM 搭載のオーケストレーターエージェントを使用して、目標の根拠を示し、サブタスクをワーカーエージェントに委任し、最終出力を合成します。

どちらのパラダイムでも、オーケストレーターは以下を実行します。

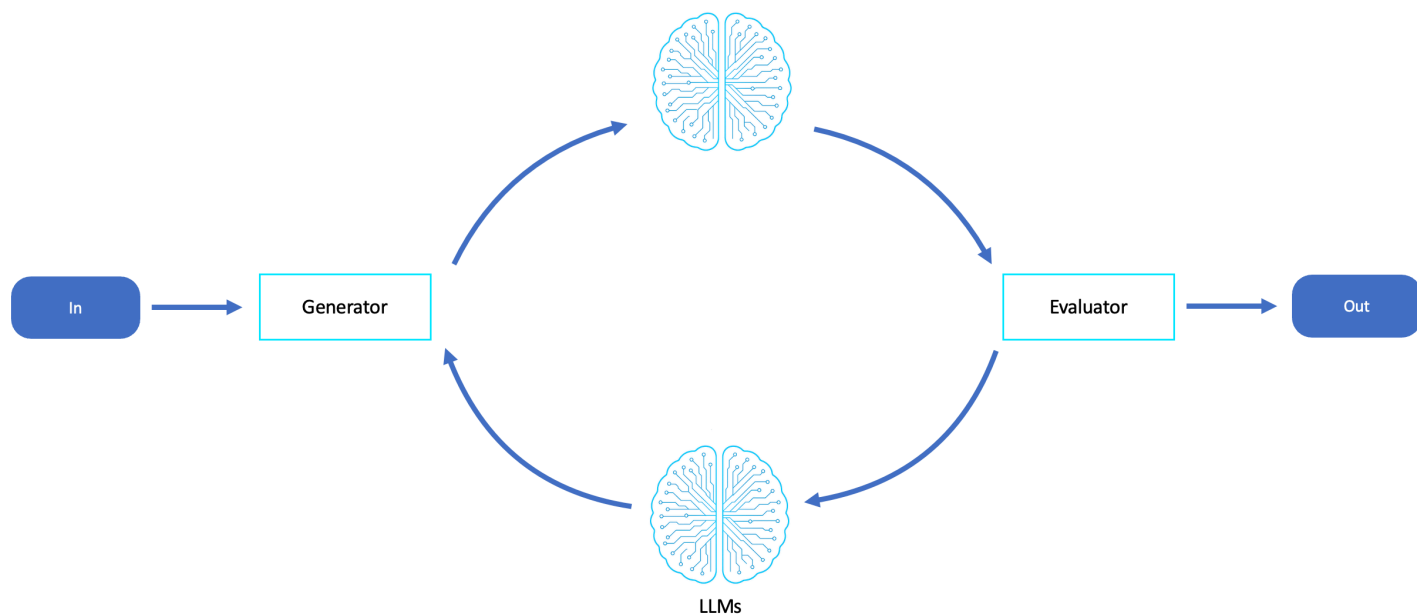
- コンテキストと実行フローを維持します
- 分岐、シーケンス、エラー処理を処理します
- 分散コンポーネントから統一された結果を生成します

ただし、エージェントオーケストレーションは推論、適応性、セマンティック委任を追加します。これにより、オープンエンド、あいまい、進化するタスクに適しています。

評価者の反射/絞り込みループパターン

コード生成、要約、自律的な意思決定などのタスクは、ランタイムフィードバックから大きなメリットを得るため、システムは監視と改良を通じて進化できます。これを運用するために、リフレクションと改良のサイクルをイベント駆動型のフィードバック制御ループとして実装できます。これは、自律的でインテリジェントなワークフローに適したシステムエンジニアリングに着想を得たパターンです。

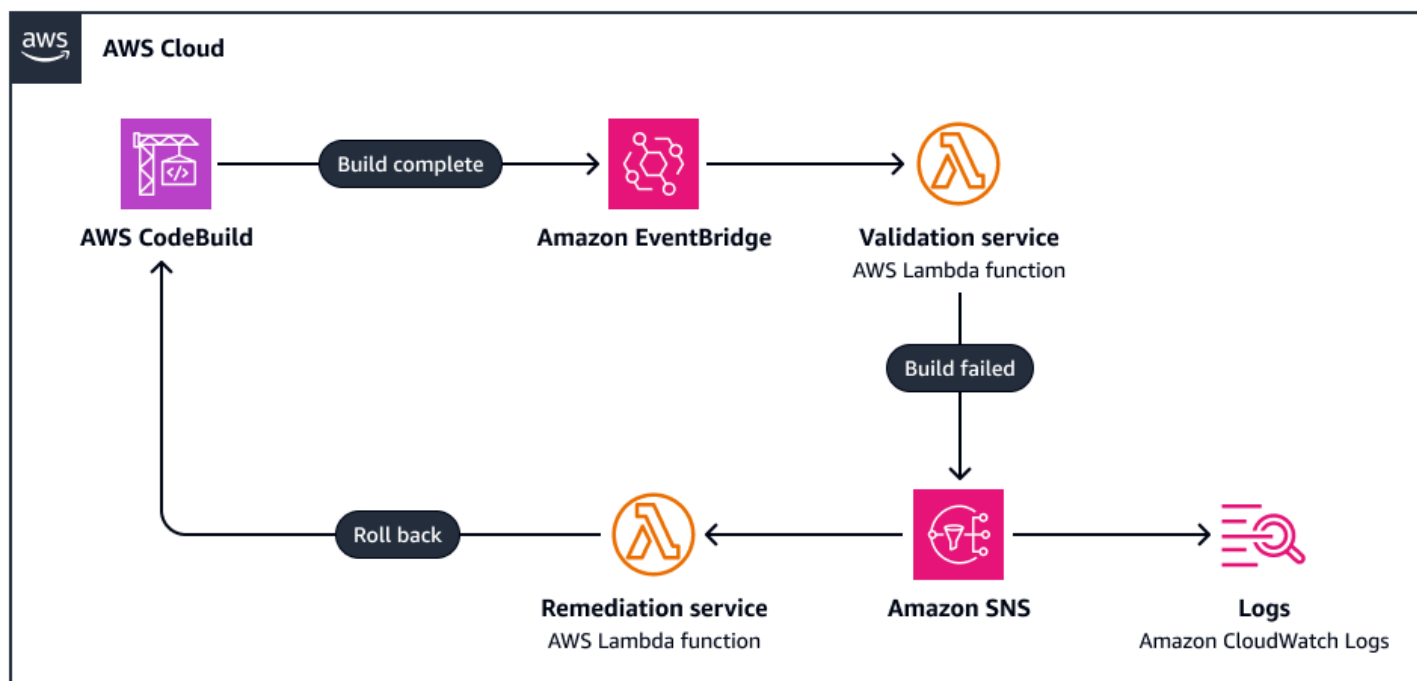
次の図は、評価者のリフレクション/絞り込みフィードバックループの例です。



フィードバックコントロールループ

フィードバックコントロールループは、独自の出力と動作をモニタリングし、定義された基準または目的の状態に照らして評価し、それに応じてアクションを調整するパターンです。このアーキテクチャは制御理論に着想を得ており、自動化、継続的インテグレーションと継続的デリバリー (CI/CD) パイプライン、機械学習オペレーションなどの分野で基盤となっています。

次の図は、フィードバックコントロールループの例です。



1. デプロイパイプラインは buildComplete イベントを出力します。
2. イベントは、ビルドを検証する自動テストジョブまたは評価ジョブをトリガーします。
3. 検証が失敗した場合 (テストの失敗、セキュリティの問題、ポリシー違反など)、システムは次の操作を行います。
 - buildComplete イベントを発行する
 - 問題をログに記録するか、通知を送信します
 - ロールバック、パッチ適用、再試行などの修復または修正アクションをトリガーします

ループは、許容可能な結果またはエスカレーションを生成するか、タイムアウトが発生するまで続行されます。このパターンは、一般的に以下に使用されます。

- 評価または修復タスクにイベントをルーティングする Amazon EventBridge ルール
- AWS Step Functions 反復再試行ロジックと評価結果の分岐用
- フィードバックトリガーとアラートの Amazon Simple Notification Service (Amazon SNS) または Amazon CloudWatch アラーム
- AWS Lambda 是正措置を適用するための 関数またはコンテナ化されたワーカー

フィードバックコントロールループ (評価者)

評価者ワークフローは、LLMsまたは推論エージェントを搭載した認知フィードバックループです。このプロセスは、以下で構成されます。

1. ジェネレーターエージェントまたは LLM は出力 (計画、回答、ドラフトなど) を生成します。
2. 評価者エージェントは、批評プロンプトまたは評価ループリックを使用して結果を確認します。
3. フィードバックに基づいて、元のエージェントまたは新しいオプティマイザエージェントが出力を修正します。

ループは、結果が一連の基準を満たすか、承認されるか、再試行制限に達するまで繰り返されます。

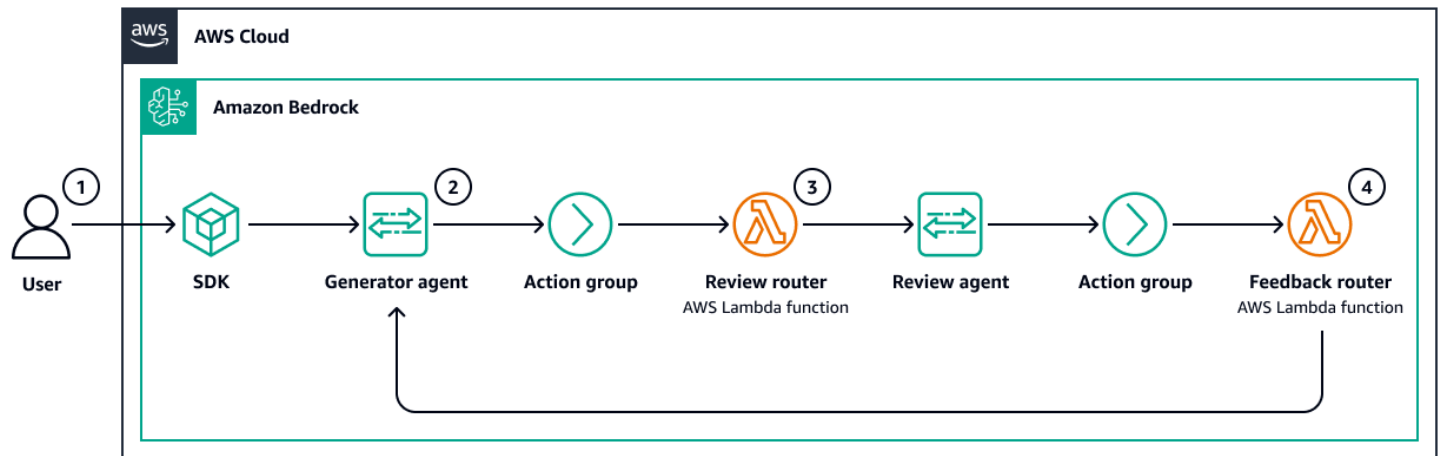
評価者

1. ユーザーは、ポリシーの概要を書き込むようにエージェントに依頼します。
2. ジェネレーターエージェントはそれをドラフトします。
3. 評価者エージェントは、カバレッジ、トーン、法的正確性をチェックします。

4. レスポンスが不十分な場合は、フィードバックループが収束するまで繰り返され、再送信されます。

これにより、自己評価、反復的な繰り返し、適応的な出力制御がすべて人間による入力なしで可能になります。

次の図は、フィードバックコントロールループ (評価者) の例です。



1. ユーザーがタスクを発行する (ビジネス戦略のドラフトを作成するなど)。
2. Amazon Bedrock エージェントは、LLM を使用して最初のドラフトを生成します。
3. 2 番目のエージェント (またはフォローアッププロンプト) は構造化された評価 (たとえば、「この出力を明確さ、完全性、トーンで評価する」) を実行します。
4. 評価がしきい値を下回ると、レスポンスは次のように修正されます。
 - 批評が埋め込まれたジェネレーターの再呼び出し
 - 特殊なリファイナーエージェントにフィードバックを送信する
 - 許容可能な応答に達するまで反復する

AWS Lambda コントローラーや などのオプションコンポーネントは、フィードバックのしきい値、再試行、フォールバック戦略を管理 AWS Step Functions できます。

重要事項

従来のフィードバックコントロールループがイベント、メトリクス、修復ロジックを使用してシステム動作を検証および調整する場合、エージェント評価者ループは推論エージェントを使用して出力を動的に評価、反映、修正します。

両方のパラダイムで:

- 出力は生成後に評価されます
- 修正アクションまたは改良アクションは、フィードバックに基づいてトリガーされます
- システムがターゲットの品質または目標に継続的に適応する

エージェントバージョンは静的検証をセマンティックリフレクションに変換し、独自の有効性を評価する自己改善エージェントを可能にします。

でのエージェントワークフローの設計 AWS

このガイドの各パターンは、を使用して構築できます AWS のサービス。Amazon Bedrock エージェントは、オーケストレーション、データアクセス、インタラクションチャネルを提供します。

コンポーネント	AWS のサービス	目的
LLM の推論	Amazon Bedrock	エージェントロジック、計画、ツールの使用
ツールの実行	AWS Lambda、Amazon ECS、Amazon SageMaker	エージェント用の外部ツールをホストする
メモリと RAG	Amazon Bedrock ナレッジベース、Amazon S3、OpenSearch	永続メモリとセマンティックメモリ
オーケストレーション	AWS Step Functions	マルチステップタスクとエージェントの調整
イベントルーティング	Amazon EventBridge、Amazon SQS	分離されたエージェント間メッセージング
[ユーザーインターフェイス]	Amazon API Gateway、AWS AppSync、SDK	アプリケーションまたはシステムのエントリポイント
モニタリング	Amazon CloudWatch、AWS X-Ray AWS 、Distro for OpenTelemetry	オブザーバビリティとエージェントのイントロスペクション

結論

エージェントワークフローパターンは、イベント駆動型アーキテクチャの次の進化段階であり、ビジネスロジックは静的に定義されず、大規模言語モデル (LLM) で強化された認識を使用して動的に推論されます。従来のクラウドネイティブプリミティブを LLM ワークフローやエージェント設計パターンと組み合わせることで、組織は目的に応じて対応し、経験から学ぶ適応型、インテリジェント、モジュラーシステムを構築できます。

これらのパターンでは、Amazon Bedrock はエージェント認識へのゲートウェイであり、LLM ベースのエージェントがイベントワークフローにアクセスし、ツールとメモリを操作し、構造化され、追跡可能で、調整された結果を提供できるようにします。

エージェントシステムを設計およびデプロイする際、これらのワークフローパターンは、自律的で構成可能な AI アーキテクチャを構築するための設計図を提供します。これらのシステムはサーバーレスのベストプラクティスに基づいており、インテリジェントな基盤モデルで強化されています。

ドキュメント履歴

以下の表は、本ガイドの重要な変更点について説明したものです。今後の更新に関する通知を受け取る場合は、[RSS フィード](#) をサブスクライブできます。

変更	説明	日付
初版発行	—	2025 年 7 月 14 日

翻訳は機械翻訳により提供されています。提供された翻訳内容と英語版の間で齟齬、不一致または矛盾がある場合、英語版が優先します。