



의 에이전트 AI 패턴 및 워크플로 AWS

AWS 권장 가이드



AWS 권장 가이드: 의 에이전트 AI 패턴 및 워크플로 AWS

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

Amazon의 상표 및 트레이드 드레스는 Amazon 외 제품 또는 서비스와 함께, Amazon 브랜드 이미지를 떨어뜨리거나 고객에게 혼동을 일으킬 수 있는 방식으로 사용할 수 없습니다. Amazon이 소유하지 않은 기타 모든 상표는 Amazon과 제휴 관계이거나 관련이 있거나 후원 관계와 관계없이 해당 소유자의 자산입니다.

Table of Contents

소개	1
대상 독자	1
목표	1
이 콘텐츠 시리즈 정보	2
에이전트 패턴	3
기본 추론 에이전트	4
아키텍처	4
설명	5
기능	6
제한 사항	6
일반 사용 사례	6
구현 지침	6
요약	7
아키텍처	7
설명	8
기능	9
일반 사용 사례	9
구현 지침	9
요약	10
함수 호출을 위한 도구 기반 에이전트	10
아키텍처	10
설명	11
기능	12
일반 사용 사례	12
구현 지침	12
요약	13
서버용 도구 기반 에이전트	13
아키텍처	13
설명	14
기능	15
일반 사용 사례	15
구현 지침	15
요약	16
컴퓨터 사용 에이전트	16

아키텍처	16
설명	17
기능	18
일반 사용 사례	18
구현 지침	18
요약	19
코딩 에이전트	19
아키텍처	19
설명	20
기능	21
일반 사용 사례	21
구현 지침	22
요약	22
음성 및 음성 에이전트	22
아키텍처	22
설명	23
기능	24
일반 사용 사례	24
구현 지침	24
요약	25
워크플로 오케스트레이션 에이전트	25
아키텍처	25
설명	26
기능	27
일반 사용 사례	27
구현 지침	27
요약	28
메모리 증강 에이전트	28
아키텍처	28
설명	29
기능	30
일반 사용 사례	30
메모리 증강 에이전트 구현	30
메모리 주입 프롬프트 구현	31
요약	31
시뮬레이션 및 테스트베드 에이전트	32

아키텍처	32
설명	32
기능	33
일반 사용 사례	34
구현 지침	34
요약	35
관찰자 및 모니터링 에이전트	35
아키텍처	36
설명	36
기능	37
일반 사용 사례	37
구현 지침	37
요약	38
다중 에이전트 공동 작업	38
설명	40
기능	40
일반 사용 사례	40
구현 지침	41
요약	41
결론	42
요점	42
LLM 워크플로	43
LLM 중강 인식 개요	43
프롬프트 체인을 위한 워크플로	44
설명	45
기능	45
일반 사용 사례	46
라우팅을 위한 워크플로	46
기능	47
일반 사용 사례	47
병렬화를 위한 워크플로	47
기능	49
일반 사용 사례	49
오케스트레이션을 위한 워크플로	49
기능	50
일반 사용 사례	50

평가자 및 반사 구체화 루프를 위한 워크플로	50
일반 사용 사례	51
기능	52
결론	52
에이전트 워크플로 패턴	53
이벤트 기반 시스템에서 인지 증강 시스템으로	53
이벤트 중심 아키텍처	53
Cognition 증강 워크플로	54
핵심 인사이트	56
프롬프트 체인 Saga 패턴	56
Saga 코레오그래피	56
프롬프트 체인 패턴	57
에이전트 코레오그래피	57
요점	59
동적 디스패치 패턴 라우팅	59
동적 디스패치	60
LLM 기반 라우팅	61
에이전트 라우터	62
요점	63
병렬화 및 산점 수집 패턴	63
확산 및 수집	64
LLM 기반 병렬화(산점 수집 인식)	66
에이전트 병렬화	66
요점	67
Saga 오케스트레이션 패턴	67
이벤트 오케스트레이션	68
역할 기반 에이전트 시스템(오케스트레이터)	69
Supervisor	69
요점	71
평가자 반사 구체화 루프 패턴	71
피드백 제어 루프	72
피드백 제어 루프(평가자)	73
평가자	73
요점	74
에서 에이전트 워크플로 설계 AWS	74
결론	75

문서 기록 76

..... lxxvii

의 에이전트 AI 패턴 및 워크플로 AWS

Aaron Sempf 및 Andrew Hooker, Amazon Web Services

2025년 7월([문서 기록](#))

조직은 대규모 언어 모델(LLMs)과 소프트웨어 에이전트를 채택하여 에이전트 패턴이라는 새로운 아키텍처 원칙을 사용하여 동적 다중 도메인 문제를 해결하고 있습니다. 에이전트 패턴은 여러 컨텍스트에서 목표 지향 AI 에이전트를 설계하고 오케스트레이션하는 데 사용되는 기본 청사진 및 모듈식 구문입니다.

대상 독자

이 가이드는 정적 로직, 심볼 로직 및 결정적 자동화를 넘어 지능형 애플리케이션을 구축하려는 아키텍트, 개발자 및 제품 리더를 대상으로 합니다.

목표

이 가이드는 제어 가능하고 목표에 맞게 조정하면서 자율적으로 작동하는 AI 에이전트 시스템에 대한 설계 프레임워크 및 구현 접근 방식을 제공합니다. 이벤트 기반 아키텍처 패턴을 다양한 에이전트 대안과 연결하여 클라우드 네이티브 아키텍처를 사용하여 프로덕션급 에이전트 시스템을 구축하는 방법을 보여줍니다. 이 가이드에서는 다음 주제에 대해 설명합니다.

- **에이전트 패턴** - 에이전트 패턴은 개별 에이전트의 구조와 동작을 설명하는 재사용 가능한 설계 템플릿입니다. 여기에는 추론 에이전트, 검색 증강 에이전트, 코딩 에이전트, 음성 인터페이스, 워크플로 오케스트레이터 및 협업 다중 에이전트 시스템이 포함됩니다. 각 패턴은 에이전트가 어떻게 인지하고, 추론하고, 행동하고, 학습하고, 매핑되는지 보여줍니다 AWS 서비스.
- **LLM 워크플로** - 워크플로는 에이전트가 추론을 위해 LLMs 사용하는 방법에 중점을 둡니다. 프롬프트 전략과 계획 메커니즘을 살펴보고 LLMs 사용하여 텍스트를 생성할 뿐만 아니라 에이전트 루프 내에서 구조화되고 해석 가능하며 신뢰할 수 있는 동작을 유도하는 방법을 간략하게 설명합니다.
- **에이전트 워크플로 패턴** - 워크플로 패턴은 여러 에이전트, 도구 및 환경이 상호 작용하여 자율 시스템을 형성하는 방법을 설명합니다. 여기에는 작업 오케스트레이션, 하위 에이전트 위임, 이벤트 기반 조정, 관찰성 및 제어에 대한 패턴이 포함됩니다. 이러한 측면은 확장 가능하고 구성 가능하며 감사 가능한 AI 아키텍처를 촉진합니다.

이 콘텐츠 시리즈 정보

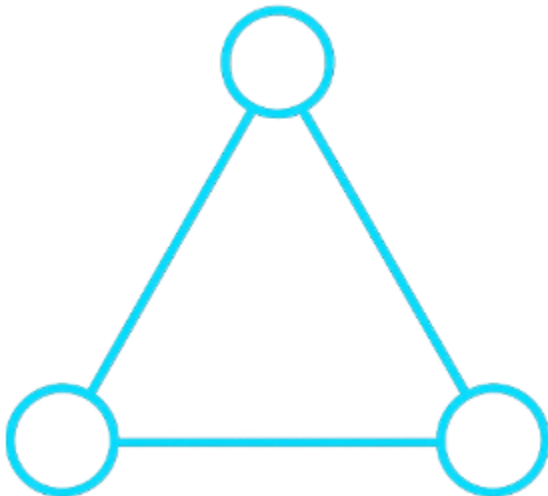
이 가이드는 에이전트 AI on에 대한 시리즈의 일부입니다 AWS. 자세한 내용과 이 시리즈의 다른 가이드를 보려면 AWS 권장 가이드 웹 사이트의 [에이전트 AI](#)를 참조하세요.

에이전트 패턴

에이전트 패턴은 특정 도메인, 사용 사례 및 복잡성 수준에 맞게 조정할 수 있는 재사용 가능하고 구성 가능한 구성 요소입니다. 그러나 에이전트 시스템은 기존 애플리케이션과 다릅니다. 모든 AI 에이전트 설계의 핵심은 다음 세 가지 기본 원칙을 기반으로 하는 개념 모델입니다.

- 비동기 - 에이전트는 느슨하게 결합되고 이벤트가 풍부한 환경에서 작동합니다.
- 자율성 - 에이전트는 인간 또는 외부 제어 없이 독립적으로 행동합니다.
- 에이전트 - 에이전트는 사용자 또는 시스템을 대신하여 특정 목표를 향해 의도적으로 행동합니다.

다음 다이어그램의 삼각형은 소프트웨어 에이전트의 핵심 구성 요소인 지각, 이유 및 작업을 나타냅니다. 이를 통해 에이전트 시스템은 환경 내에서 관찰하고, 결정을 내리고, 조치를 취할 수 있습니다.



설계상 에이전트 패턴은 AI 시스템을 구축하기 위한 모듈식 설계 언어를 제공하므로 액세스, 운영, 확장 및 프로덕션 준비가 가능합니다. 이러한 시스템을 설계하려면 다음 세 가지 상호 관련된 차원에 주의해야 하며, 이에 대해서는 이 가이드의 뒷부분에서 자세히 설명합니다.

이 섹션의 내용

- [기본 추론 에이전트](#)
- [함수 호출을 위한 도구 기반 에이전트](#)
- [서버용 도구 기반 에이전트](#)
- [컴퓨터 사용 에이전트](#)
- [코딩 에이전트](#)

- [음성 및 음성 에이전트](#)
- [워크플로 오케스트레이션 에이전트](#)
- [메모리 증강 에이전트](#)
- [시뮬레이션 및 테스트베드 에이전트](#)
- [관찰자 및 모니터링 에이전트](#)
- [다중 에이전트 공동 작업](#)

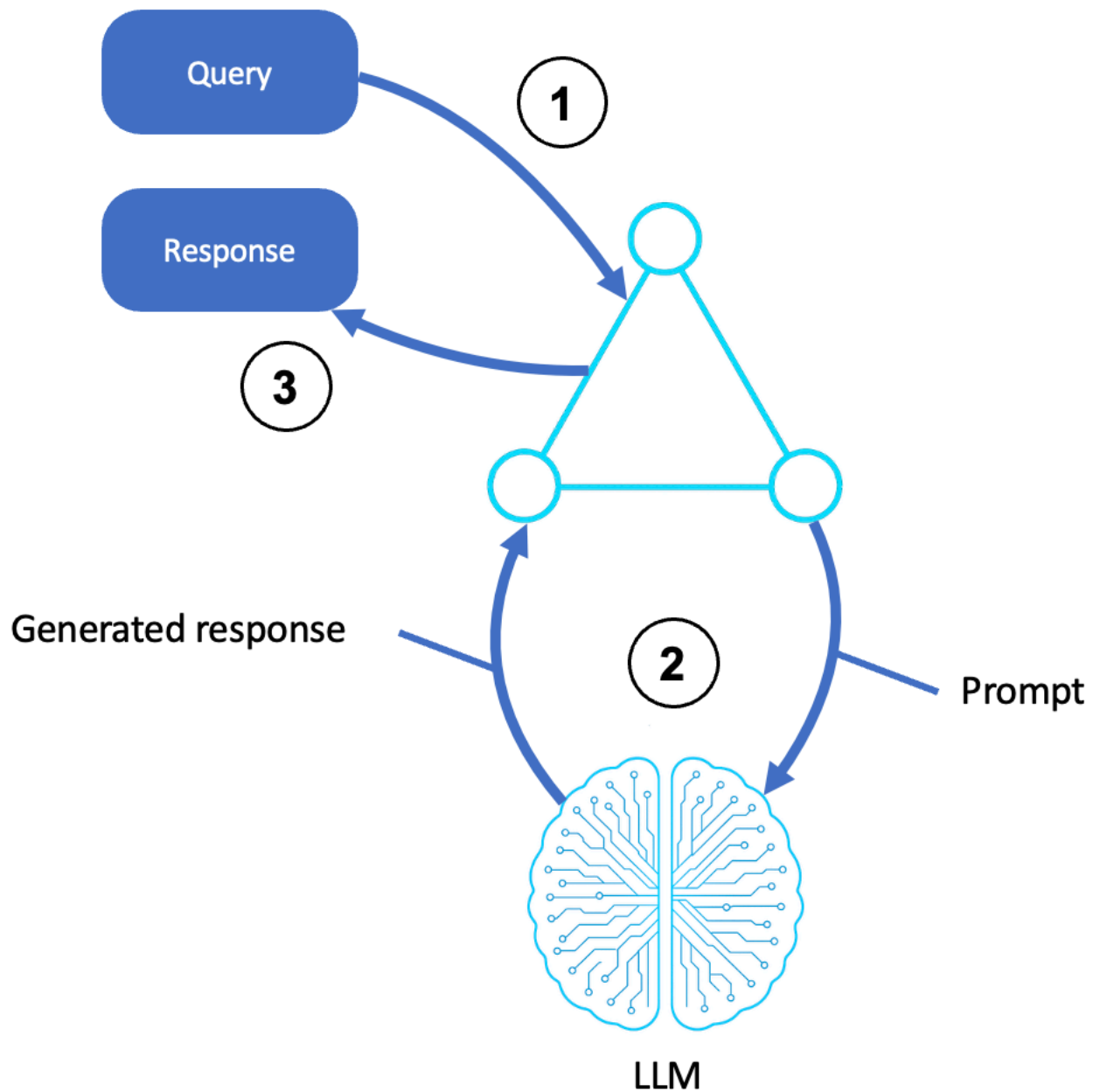
기본 추론 에이전트

기본 추론 에이전트는 쿼리에 대한 응답으로 논리적 추론 또는 의사 결정을 수행하는 가장 간단한 형태의 에이전트 AI입니다. 사용자 또는 시스템의 입력을 수락하고 쿼리를 처리하고 구조화된 프롬프트를 사용하여 응답을 생성합니다.

이 패턴은 주어진 컨텍스트를 기반으로 한 단일 단계 추론, 분류 또는 요약이 필요한 작업에 유용합니다. 메모리, 도구 또는 상태 관리를 사용하지 않으므로 대규모 워크플로에서 상태 비저장, 경량 및 고도로 구성 가능합니다.

아키텍처

기본 추론 에이전트의 흐름은 다음 다이어그램에 나와 있습니다.



설명

1. 입력을 수신합니다.

- 사용자, 시스템 또는 업스트림 에이전트가 쿼리 또는 명령을 제출합니다.
- 입력은 에이전트 셸 또는 오케스트레이션 계층으로 전달됩니다.
- 이 단계에는 모든 사전 처리, 프롬프트 템플릿 지정 및 목표 식별이 포함됩니다.

2. LLM을 호출합니다.

- 에이전트는 쿼리를 구조화된 프롬프트로 변환하여 LLM으로 보냅니다(예: Amazon Bedrock을 통해).
 - LLM은 사전 훈련된 지식 및 컨텍스트를 사용하여 프롬프트를 기반으로 응답을 생성합니다.
 - 생성된 출력에는 추론 단계(chain-of-thought), 최종 답변 또는 순위 옵션이 포함될 수 있습니다.
3. 응답을 반환합니다.
- 생성된 출력은 에이전트의 인터페이스로 릴레이됩니다.
 - 여기에는 형식 지정, 사후 처리 또는 API 응답이 포함될 수 있습니다.

기능

- 자연어 또는 구조화된 입력 지원
- 프롬프트 엔지니어링을 사용하여 동작을 안내합니다.
- 상태 비저장 및 확장 가능
- UI, CLI, APIs 및 파이프라인에 포함할 수 있습니다.

제한 사항

- 메모리 또는 기록 인식 없음
- 외부 도구 또는 데이터 소스와의 상호 작용 없음
- 추론 시 LLM이 알고 있는 내용으로 제한됨

일반 사용 사례

- 대화 질문 및 답변
- 정책 설명 및 요약
- 의사 결정을 위한 지침
- 가볍고 자동화된 챗봇 흐름
- 분류, 레이블 지정 및 점수

구현 지침

다음 도구 및 서비스를 사용하여 기본 추론 에이전트를 생성할 수 있습니다.

- LLM 호출용 Amazon Bedrock(Anthropic, AI21, Meta)
- Amazon API Gateway 또는 AWS Lambda - 상태 비저장 마이크로서비스로 노출
- 파라미터 스토어 또는 코드 AWS Secrets Manager로 저장된 프롬프트 템플릿

요약

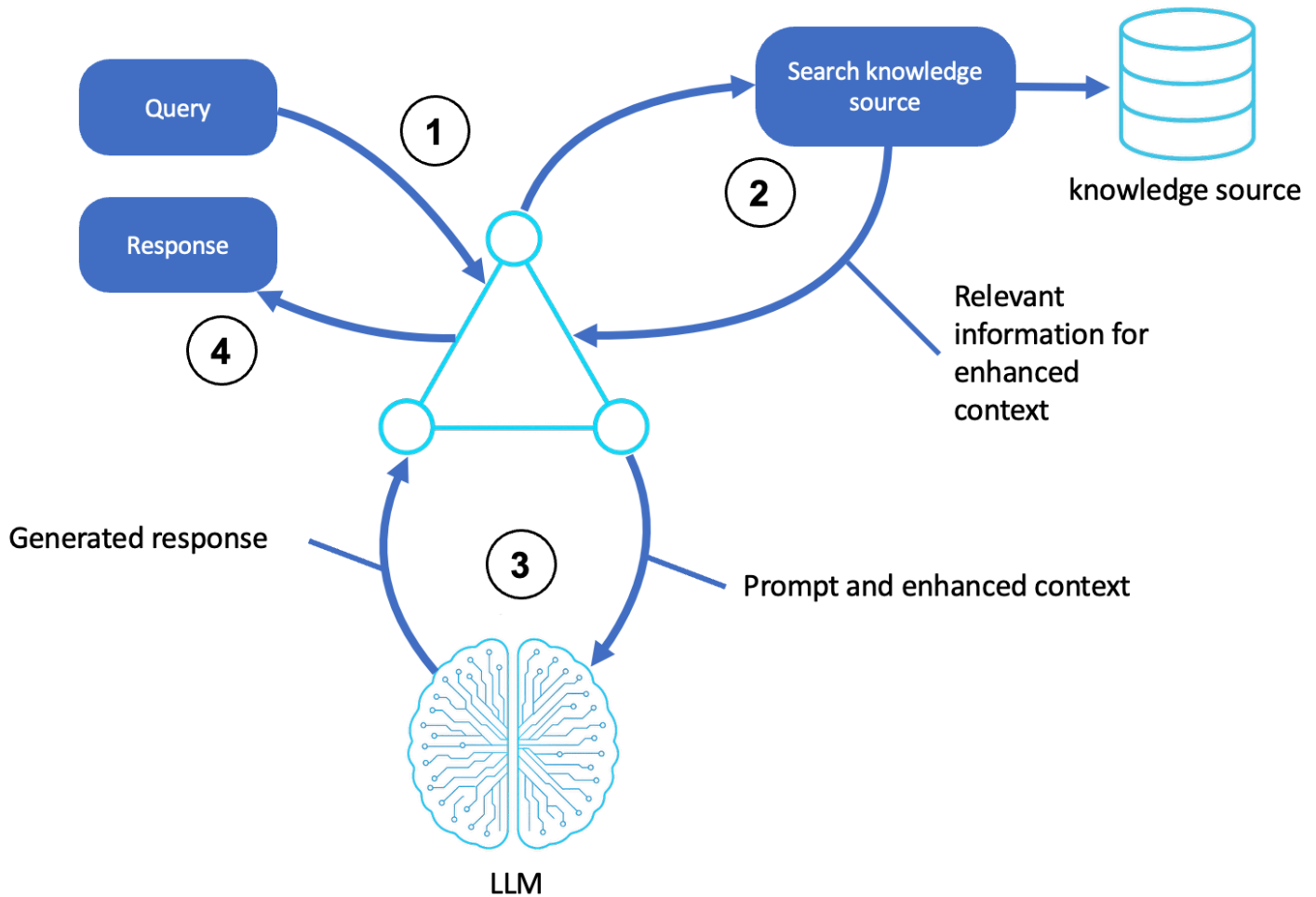
기본 추론 에이전트는 단순한 구조로 인해 기본적입니다. 목표를 지능형 출력으로 이어지는 추론 경로로 전환하는 핵심 기능이 있습니다. 이 패턴은 도구 기반 에이전트 및 검색 증강 생성(RAG)을 사용하는 에이전트와 같은 고급 패턴의 시작점인 경우가 많습니다. 또한 대규모 워크플로의 신뢰할 수 있는 모듈식 구성 요소입니다.

에이전트 RAG

검색 증강 생성(RAG)은 정보 검색과 텍스트 생성을 결합하여 정확하고 상황에 맞는 응답을 생성하는 기법입니다. RAG를 사용하면 에이전트가 LLM을 사용하기 전에 관련 외부 정보를 검색할 수 있습니다. 최신 정보, up-to-date 사실 정보 또는 도메인별 정보를 바탕으로 에이전트의 효과적인 메모리와 추론 정확도를 높입니다. 사전 훈련된 가중치에만 의존하는 상태 비저장 LLMs과 달리 RAG에는 컨텍스트를 사용하여 프롬프트를 동적으로 개선하는 외부 지식 검색 계층이 있습니다.

아키텍처

RAG 패턴의 로직은 다음 다이어그램에 나와 있습니다.



설명

1. 쿼리를 수신합니다.

- 사용자 또는 업스트림 시스템은 에이전트에게 쿼리 또는 목표를 제출합니다.
- 에이전트 셸은 요청을 수락하고 추론을 위한 프롬프트로 형식을 지정합니다.

2. 외부 소스를 검색합니다.

- 에이전트는 쿼리에서 개념과 의도를 식별합니다.
- 의미 체계 검색 또는 키워드 일치를 사용하여 벡터 저장소, 데이터베이스 또는 문서 인덱스와 같은 지식 소스를 쿼리합니다.
- 가장 관련성이 높은 구절, 문서 또는 엔터티는 다음 단계에서 사용할 수 있도록 검색됩니다.

3. 컨텍스트 응답을 생성합니다.

- 에이전트는 검색된 정보로 프롬프트를 보강하여 LLM에 대한 컨텍스트 강화 입력을 형성합니다.

- LLM은 생성형 추론(예: chain-of-thought 또는 반영)을 사용하여 모든 입력을 처리하여 정확한 응답을 생성합니다.

4. 최종 출력을 반환합니다.

- 에이전트는 출력을 모든 통신 헤더 또는 필요한 서식으로 래핑하여 준비한 다음 사용자 또는 호출 시스템에 반환합니다.
- (선택 사항) 검색된 문서 및 LLM 출력은 향후 쿼리를 위해 로깅되고 점수가 매겨지고 메모리에 저장될 수 있습니다.

기능

- 롱테일 또는 엔터프라이즈별 도메인에서도 사실 기반 출력
- 모델을 미세 조정하지 않는 메모리 확장
- 각 쿼리 및 사용자 상태에 따른 동적 컨텍스트
- 벡터 데이터베이스, 의미 체계 인덱스 및 메타데이터 필터링과 완벽하게 호환

일반 사용 사례

- 엔터프라이즈 지식 도우미
- 규정 준수 봇
- 고객 지원 부조종사
- 검색 강화 챗봇
- 개발자 설명서 에이전트

구현 지침

다음 도구 및 서비스를 사용하여 RAG를 사용하는 에이전트를 생성합니다.

- LLM 호출을 위한 Amazon Bedrock
- 설명서 또는 구조화된 데이터 검색을 위한 Amazon Kendra, OpenSearch 또는 Amazon Aurora
- 문서 스토리지용 Amazon Simple Storage Service(Amazon S3)
- AWS Lambda 검색, 프롬프트 및 LLM 추론을 오케스트레이션하려면
- 에이전트와의 지식 기반 통합(메모리 플러그인, 의미 체계 리트리버 또는 Amazon Bedrock 사용)

요약

에이전트 RAG는 정적 모델 추론을 동적 실제 인텔리전스에 연결합니다. 에이전트가 모르는 내용을 조회하고, 검색된 지식에서 답변을 합성하고, 신뢰도가 높고 감사 가능한 응답을 생성할 수 있는 기능을 제공합니다.

RAG 패턴은 재훈련 없이 지식 액세스를 확장하는 지능형 에이전트를 구축하는 기반입니다. 도구 사용, 계획 및 장기 메모리와 관련된 보다 복잡한 오케스트레이션 패턴의 전조인 경우가 많습니다.

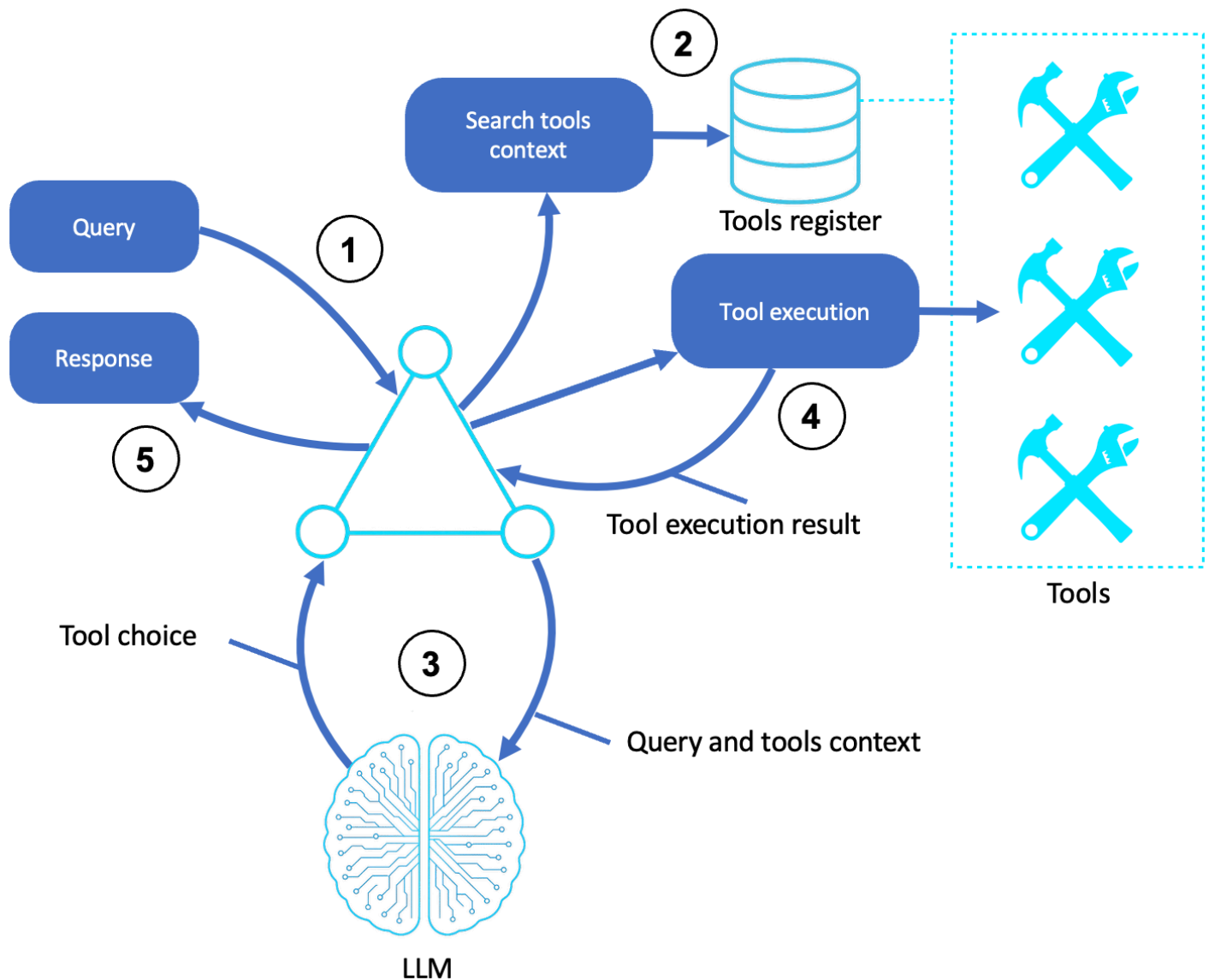
함수 호출을 위한 도구 기반 에이전트

도구 기반 에이전트는 언어 전용 추론을 넘어서는 작업을 완료하기 위해 외부 함수 또는 APIs 호출하여 에이전트 추론 기능을 확장합니다. 이 패턴은 LLM을 사용하여 사용할 도구를 결정한 다음 호출 인수를 생성하고 도구의 출력을 추론 루프에 통합합니다.

이 패턴을 사용하면 에이전트가 단순히 응답을 제공하는 대신 조치를 취할 수 있습니다. 도구 인터페이스는 산술 계산 및 데이터베이스 조회부터 외부 APIs 및 클라우드 서비스에 이르기까지 호출 가능한 모든 기능을 나타냅니다.

아키텍처

함수를 호출하기 위한 도구 기반 에이전트는 다음 다이어그램에 나와 있습니다.



설명

1. 쿼리를 수신합니다.

- 에이전트는 사용자 또는 호출 시스템으로부터 자연어 쿼리 또는 작업을 수신합니다.

2. 도구 검색

- 에이전트는 내부 메타데이터 또는 도구 레지스트리를 사용하여 사용 가능한 도구, 스키마 및 관련 기능을 검색합니다.

3. 도구를 선택하고 호출합니다.

- LLM은 프롬프트에서 쿼리 및 도구 메타데이터(예: 함수 이름, 입력 유형 및 설명)를 수신합니다.
- 가장 관련성이 높은 도구를 선택하고, 입력 인수를 구성하고, 구조화된 함수 호출을 반환합니다.

4. 선택한 도구를 실행합니다.

- 에이전트 셀 또는 도구 실행기는 선택한 함수를 실행하고 결과(예: API 출력, 데이터베이스 값 또는 계산)를 반환합니다.

5. 응답을 반환합니다.

- LLM은 직접 또는 업데이트된 프롬프트의 일부로 결과를 에이전트에 전달합니다. 그런 다음 자연어 결과를 반환합니다.

기능

- 작업 컨텍스트에 따른 동적 도구 선택
- 스키마 기반 프롬프트(OpenAPI, JSON 스키마, AWS 함수 인터페이스)
- 결과 해석 및 출력을 추론으로 연결
- 상태 비저장 또는 세션 인식 작업

일반 사용 사례

- 외부 데이터 액세스가 가능한 가상 어시스턴트
- 재무 계산기 및 예측기
- API 기반 지식 작업자
- 호출LLMs AWS Lambda, Amazon SageMaker 엔드포인트 및 SaaS 서비스

구현 지침

다음을 사용하여 함수 호출을 위한 도구 기반 에이전트를 생성합니다.

- 함수 호출을 지원하는 Amazon Bedrock(Anthropic Claude)
- AWS Lambda 도구 실행 백엔드로 사용
- Amazon API Gateway 또는 도구 오케스트레이션 AWS Step Functions 용
- 컨텍스트 인식 도구 메타데이터를 위한 Amazon DynamoDB 또는 Amazon Relational Database Service(RDS)
- 출력을 라우팅하기 위해 상태를 매핑 AWS Step Functions 하는 Amazon EventBridge 파이프라인 또는

요약

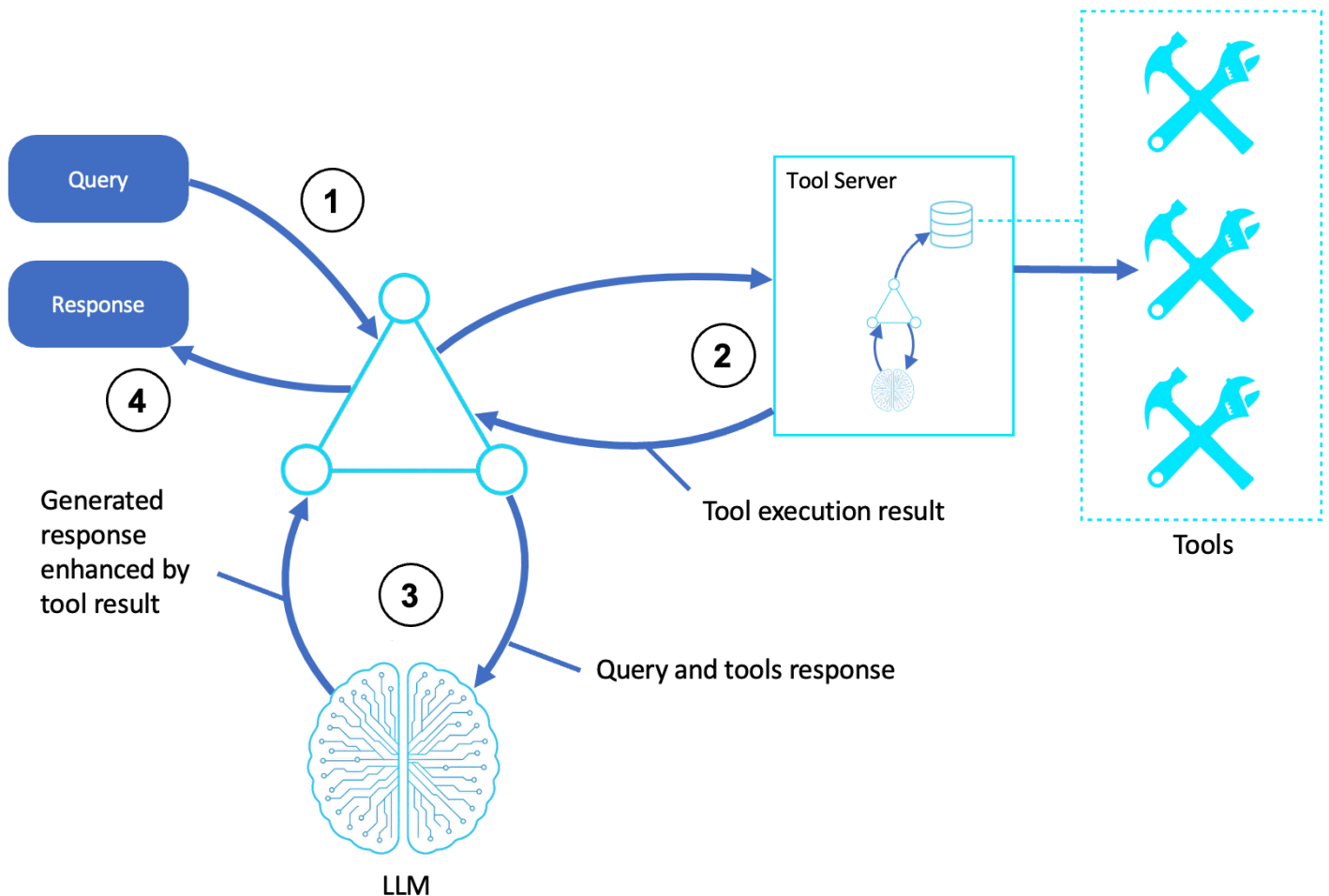
도구 기반 함수 호출 에이전트는 언어 이해에서 작업 수행으로의 전환을 나타냅니다. 이러한 에이전트는 LLM 추론을 유지하면서 동적 컨텍스트 인식 도구를 호출하여 수동 어시스턴트를 작업을 완료하고, 서비스에 액세스하고, 비즈니스 운영을 통합하는 시스템으로 변환합니다. 이 패턴은 특히 선언적 스키마, 권한 부여 프레임워크 및 다중 에이전트 시스템과 결합할 때 엔터프라이즈 설정에서 에이전트 AI의 중요한 구성 요소입니다.

서버용 도구 기반 에이전트

서버용 도구 기반 에이전트는 도구, 스크립트 및 복합 에이전트 전용 런타임 환경이 있는 외부 서버에 도구 실행을 위임하여 함수 호출 에이전트를 개선합니다. 에이전트 루프가 선택하고 호출하는 인라인 함수 호출과 달리 서버 기반 에이전트는 로직 및 실행 파이프라인을 다른 에이전트 또는 시스템에 아웃소싱합니다. 이를 통해 다중 도구 체인, 격리된 실행 및 특수 추론과 같은 고급 기능을 제공합니다. 도구 서버는 도구 자체에 별도의 AI 모델, 비즈니스 규칙 또는 환경이 포함될 수 있는 복잡하거나 상태 저장적이거나 리소스 집약적인 작업에 적합합니다.

아키텍처

다음은 서버용 도구 기반 에이전트의 패턴입니다.



설명

1. 쿼리를 수신합니다.

- 사용자 또는 시스템이 에이전트 셀에 요청을 제출합니다.
- 에이전트는 쿼리를 해석하고 도구 서버로 디스패치할 준비를 합니다.

2. 도구 서버 프로세스를 실행합니다.

- 에이전트는 구조화된 파라미터와 함께 작업을 도구 서버로 전송합니다.
- 그러면 도구 서버가 다음을 수행할 수 있습니다.
 - 전용 컴퓨팅 시스템(예: AWS Lambda컨테이너 또는 Amazon SageMaker)에서 스크립트 또는 로직 실행
 - LLM 추론과 함께 자체 하위 에이전트를 사용하여 도구 선택 및 실행
 - 종속성, 재시도 또는 다단계 실행 흐름 관리
 - 작업이 완료되면 기본 에이전트에 결과 출력

3. 도구 출력과 함께 LLM 추론 사용

- 에이전트는 LLM을 호출하여 원본 쿼리와 도구 서버 결과를 프롬프트의 일부로 전달합니다.
- LLM은 새로 획득한 정보를 통합하는 응답을 합성합니다.

4. 응답을 반환합니다.

- 에이전트는 사용자 또는 호출 시스템에 자연어 또는 구조화된 응답을 반환합니다.
- (선택 사항) 결과는 메모리 또는 감사 로그에 저장될 수 있습니다.

기능

- 기본 에이전트 실행 루프 외부에서 도구가 호출됩니다.
- 도구 실행에는 LLM 호출, 로직 체인 또는 하위 에이전트가 포함될 수 있습니다.
- 에이전트는 도구 래퍼뿐만 아니라 컨트롤러 또는 디스패처 역할을 합니다.
- 로직의 구성 가능성, 확장성 및 격리를 활성화합니다.

일반 사용 사례

- 모델 체인 오케스트레이션(예: LLM, 비전 및 코드 결합)
- AI 기반 자동화 파이프라인
- 스크립트 실행기가 있는 DevOps 어시스턴트 에이전트
- 복잡한 재무 계산, 시뮬레이션 또는 최적화 에이전트
- 멀티모달 도구(예: 오디오, 설명서 및 작업 결합)

구현 지침

AWS 서비스다음을 사용하여이 패턴을 빌드할 수 있습니다.

- Amazon Bedrock(에이전트 호스트 및 LLM 추론)
- AWS Lambda AWS Fargate, Amazon ECS 또는 Amazon SageMaker 엔드포인트를 도구 서버 런타임으로 사용
- Amazon API Gateway 또는 AWS App Runner - 도구 서버 APIs 노출
- 분리된 agent-to-tool 메시지를 위한 Amazon EventBridge
- AWS Step Functions 도구 서버에 다중 에이전트 로직을 구성하는 AWS AppFabric 경우 또는

요약

서버를 사용하는 도구 기반 에이전트는 모듈식이며 확장 가능합니다. 결정 로직을 실행과 분리하므로 기본 에이전트는 복잡하거나 민감한 작업을 다른 시스템으로 오프로드하면서 경량 상태를 유지할 수 있습니다. 이는 엔터프라이즈급 에이전트 AI, 특히 거버넌스, 관찰성, 격리, 동적 구성 또는 이들의 조합이 필요한 환경에 중요합니다.

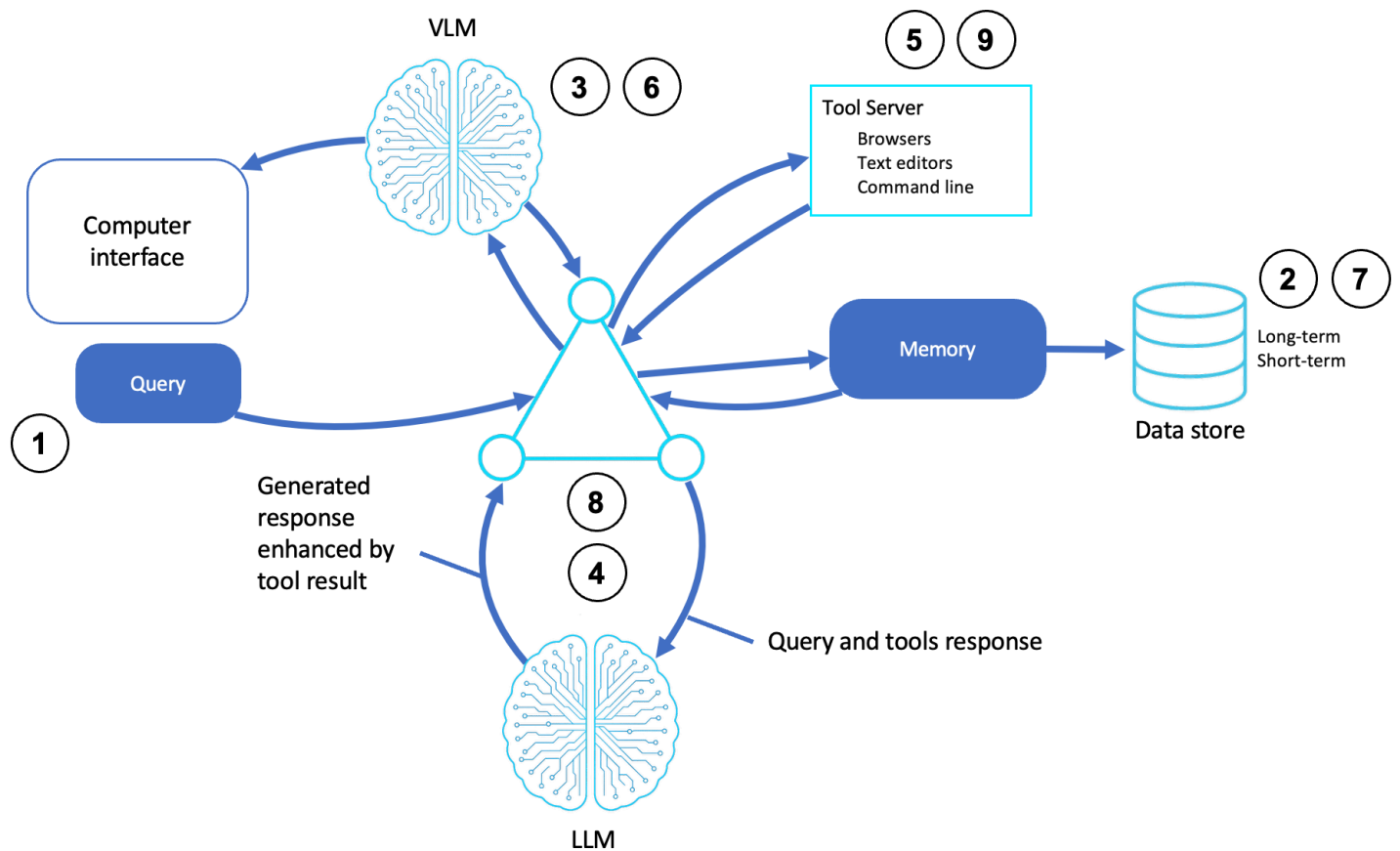
컴퓨터 사용 에이전트

컴퓨터 사용 에이전트는 브라우저, 터미널, 파일 시스템 및 애플리케이션과 같은 디지털 환경을 시뮬레이션하거나 제어할 수 있습니다. 이러한 에이전트는 LLM 추론, 시각적 언어 모델(VLMs) 및 명령을 실행하거나 입력 이벤트를 시뮬레이션하는 도구 서버를 결합하여 사용자 의도를 해석하고, 시각적 및 텍스트 인터페이스와 상호 작용하고, 목표 지향적 작업을 수행합니다.

이 패턴은 에이전트가 어시스턴트뿐만 아니라 동일한 도구와 환경을 사용하여 인간처럼 작업을 수행하는 프록시 역할을 하는 실제 AI 자동화에 중요합니다.

아키텍처

컴퓨터 사용 에이전트 패턴은 다음 다이어그램에 나와 있습니다.



설명

1. 쿼리를 수신합니다.
 - 작업 또는 요청은 UI, API 또는 자연어 인터페이스를 통해 제공됩니다.
2. 메모리에 액세스합니다.
 - 에이전트는 단기 및 장기 메모리를 검색하여 과거 명령, 목표 및 시스템 상태를 재현합니다.
3. 시각적 컨텍스트 분석
 - VLM은 컴퓨터 화면, 시스템 상태 또는 UI 요소를 관찰하여 주어진 컨텍스트를 이해하고 실행 가능한 항목을 식별합니다.
4. LLM을 통한 이유
 - LLM은 쿼리, 메모리 상태, 도구 및 서버 응답을 결합하여 다음 작업을 결정합니다.
5. 도구 서버와 상호 작용
 - 에이전트는 서버에서 호스팅되는 도구를 호출하며, 여기에는 다음이 포함될 수 있습니다.
 - 브라우저(예: 헤드리스 Chrome) 및 셸 환경
 - 텍스트 및 코드 편집기

- 사용자 지정 스크립트 인터페이스
6. 시각적 입력 업데이트
 - 시스템 UI가 변경되거나 추가 관찰이 필요한 경우 VLM은 화면 상태 또는 텍스트 버퍼를 다시 분석할 수 있습니다.
 7. 메모리 업데이트
 - 새로운 인사이트, 시스템 상태 또는 사용자 피드백은 단기 및 장기 메모리에 기록됩니다.
 8. 최종 결정 및 설명 공식화
 - LLM은 쿼리 및 도구 출력을 기반으로 결과를 합성하거나 작업을 권장합니다.
 9. 응답을 반환합니다.
 - 에이전트는 인터페이스에 결과(예: 완료된 작업, 확인 또는 생성된 콘텐츠)를 반환합니다.

기능

- 시각적 입력과 텍스트 입력을 사용한 멀티모달 추론
- 시뮬레이션된 입력 또는 API 기반 입력을 통한 애플리케이션 제어
- 영구 상태에 대한 메모리 관리
- 시퀀스 실행의 자율성(다단계 흐름)

일반 사용 사례

- IDEs에서 코드를 작성하고 실행하는 AI 개발자
- 반복적인 디지털 워크플로를 위한 컴퓨터 사용 에이전트
- 소프트웨어 테스트 및 품질 보증을 위해 시뮬레이션된 사용자
- 음성 또는 상위 수준 지침을 통해 UIs를 탐색하기 위한 접근성 에이전트
- 추론을 통해 향상된 스마트 로봇 프로세스 자동화(RPA)

구현 지침

- AWS 서비스다음을 사용하여이 패턴을 빌드할 수 있습니다.
- LLM 기반 계획 및 추론을 위한 Amazon Bedrock
- Amazon Elastic Compute Cloud(Amazon EC2) AWS Lambda또는 Amazon SageMaker 노트북으로 시뮬레이션된 UI 환경으로 도구 서버 실행

- 메모리 지속성을 위한 Amazon Simple Storage Service(Amazon S3) 또는 Amazon DynamoDB
- 하이브리드 시나리오의 UI 이미지 분석을 위한 Amazon Rekognition(또는 사용자 지정 모델)
- 관찰성 및 감사 추적을 AWS X-Ray 위한 Amazon CloudWatch Logs 또는

요약

컴퓨터 사용 에이전트는 자율 디지털 운영자 역할을 하여 인간-컴퓨터 상호 작용과 AI 기반 작업 간의 격차를 해소합니다. 이러한 에이전트는 메모리, 도구 오케스트레이션 및 VLMs 통합하여 인간을 위해 설계된 시스템과 적응형으로 상호 작용하고, 작업을 실행하고, 파일을 업데이트하고, 메뉴를 탐색하고, 응답을 생성할 수 있습니다.

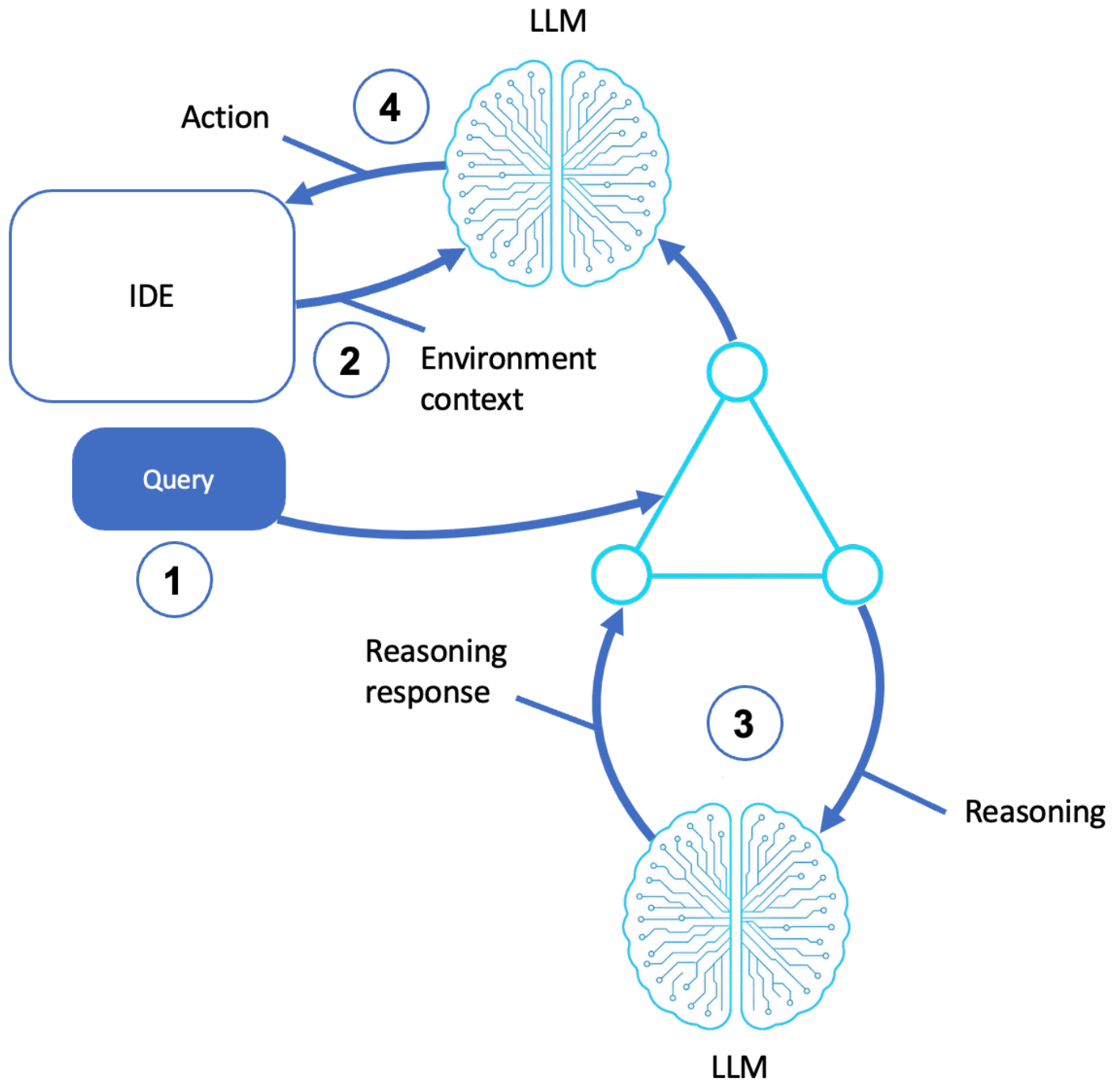
코딩 에이전트

코딩 에이전트는 프로그래밍 작업을 추론하고, 코드를 생성 또는 수정하고, IDEs 및 CLIs. 이러한 에이전트는 자연어 이해와 구조화된 추론을 결합하여 함수 생성부터 버그 수정 및 테스트 작성에 이르기까지 소프트웨어 개발을 지원, 보장 및 자동화합니다.

자동 완성 도구와 달리 코딩 에이전트는 사용자 목표를 적극적으로 해석하고, 개발 환경에서 컨텍스트를 쿼리하고(예: 파일을 열고 오류를 추적), 요구 사항을 식별한 다음 작업을 제안하고 수행합니다.

아키텍처

코딩 에이전트 패턴은 다음 다이어그램에 나와 있습니다.



설명

1. 쿼리를 수신합니다.

- 사용자는 명령 팔레트, 채팅 창 또는 CLI(예: “이 함수에 로깅 추가” 또는 “가독성을 위한 리팩터링”)를 통해 자연어 지침을 제공합니다.

2. 환경 컨텍스트 추출

- 에이전트는 활성 파일, 커서 위치, 코드 조각 및 기호 테이블을 포함하여 IDE에서 컨텍스트를 수집합니다.
- 오류 메시지, 테스트 결과 및 다른 에이전트의 출력을 출력합니다.

3. LLM 추론

- 에이전트는 쿼리 및 환경 컨텍스트를 포함한 프롬프트를 LLM으로 전송합니다.
 - LLM은 추론 패스를 수행하여 다음을 결정합니다.
 - 변경해야 할 사항
 - 솔루션을 생성하는 방법
 - 모든 리팩터링, 재작성 또는 코딩 단계

4. 작업을 실행합니다.

- LLM은 에이전트에 출력을 반환하고 IDE 또는 런타임 환경으로 가져옵니다.
- 여기에는 코드 삽입 또는 수정, 설명 또는 설명서 생성, 다운스트림 빌드, 테스트 및 린팅 작업 트리거가 포함될 수 있습니다.

기능

- 높은 컨텍스트 인식(예: IDE 상태, 커서 및 구문 트리)
- 목표 및 피드백의 반복적인 추론
- 선택적 코드 계획 및 작업 분리(예: 첫 번째 이유 후 조치)
- 동기식 또는 비동기식 개발자 워크플로에서 작동

일반 사용 사례

- 작업 설명에서 코드 생성
- 코드 리팩터링 및 최적화
- 테스트 사례 생성 및 검증
- 오류 설명 및 디버깅
- 설명서 도우미
- 페어링된 프로그래밍 코파일럿

구현 지침

- 다음 도구 및를 사용하여이 패턴을 빌드할 수 있습니다 AWS 서비스.
- LLM 기반 생성 및 추론을 위한 Amazon Bedrock
- 제안 및 완료 코딩을 위한 Amazon Q Developer
- AWS Lambda 샌드박스 환경을 실행하고 테스트하기 위한 또는 Amazon Elastic Container Service(Amazon ECS)
- AWS Cloud9, VS Code 확장 또는 컨텍스트를 호스팅하고 평가하기 위한 사용자 지정 IDE 통합
- 중간 프롬프트, 응답 및 개정 기록을 저장하기 위한 Amazon Simple Storage Service(Amazon S3)

요약

코딩 에이전트는 자연어를 해석하고, 컨텍스트를 분석하고, 다단계 코드 변경을 생성하고, 소프트웨어 개발 수명 주기와 통합할 수 있는 새로운 AI 기반 개발 도구입니다.

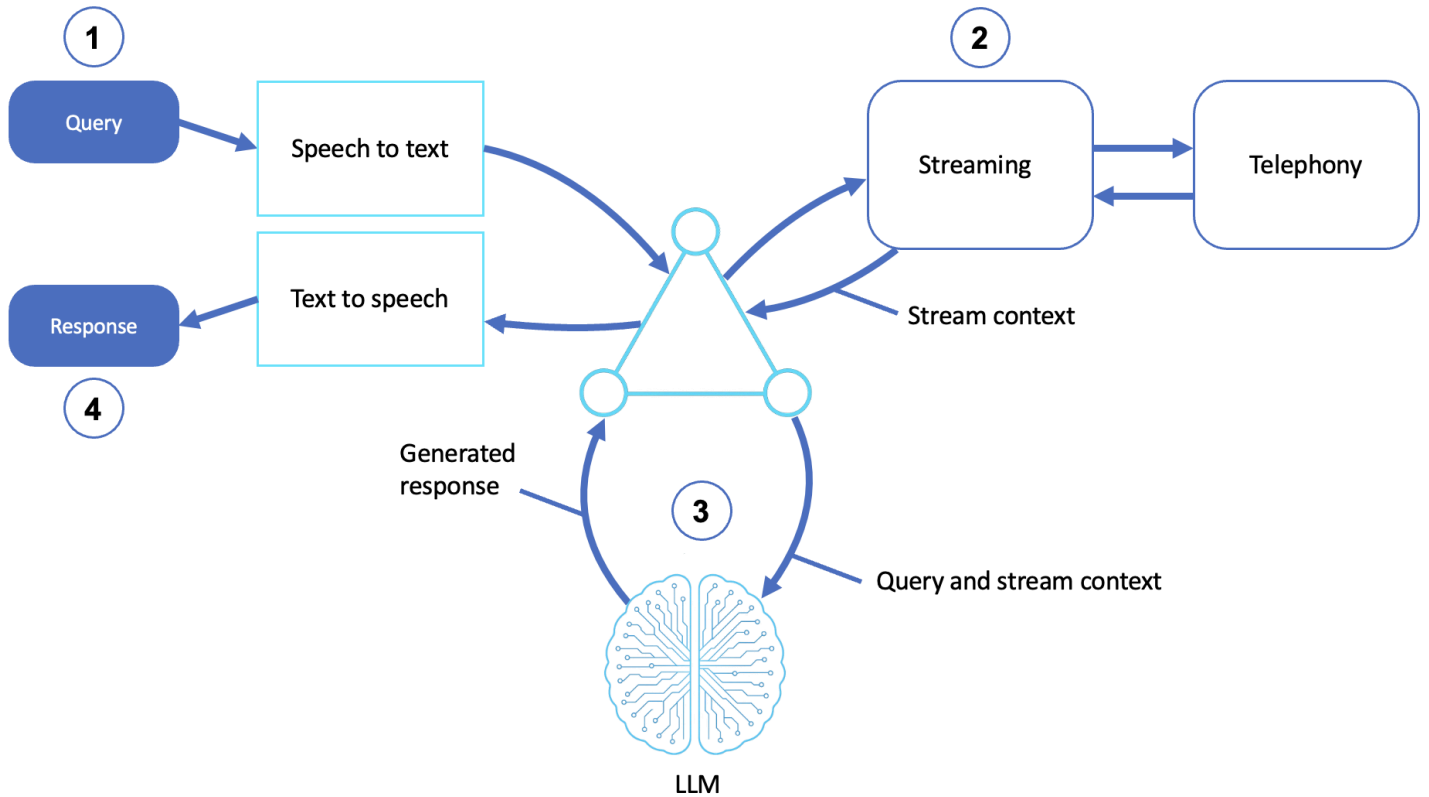
음성 및 음성 에이전트

음성 및 음성 에이전트는 음성 대화를 통해 사용자와 상호 작용합니다. 이러한 에이전트는 음성 인식, 자연어 이해 및 음성 합성을 통합하여 텔레포니, 모바일, 웹 및 임베디드 플랫폼에서 대화형 AI를 지원합니다.

음성 에이전트는 핸드프리, 실시간 또는 접근성 기반 환경에서 특히 효과적입니다. 스트리밍 인터페이스와 LLM 기반 추론을 결합하면 사용자에게 자연스럽게 느껴지는 풍부하고 동적인 상호 작용을 촉진할 수 있습니다.

아키텍처

음성 및 음성 에이전트는 다음 다이어그램에 나와 있습니다.



설명

1. 음성 쿼리를 수신합니다.

- 사용자가 전화, 마이크 또는 임베디드 시스템에 대한 요청을 음성으로 보냅니다.
- 스피|speech-to-text(STT) 모듈은 오디오를 텍스트로 변환합니다.

2. 스트리밍 및 텔레포니 컨텍스트 통합

- 에이전트는 스트리밍 인터페이스를 사용하여 오디오 I/O를 실시간으로 관리합니다.
- 고객 센터 또는 통신 컨텍스트에 배포된 경우 텔레포니 통합은 세션 라우팅, 듀얼 톤 다중 주파수 (DTMF) 입력 및 미디어 전송을 처리합니다.

참고: DTMF는 전화 키패드의 버튼을 누를 때 생성되는 톤을 나타냅니다. 음성 에이전트 내 스트리밍 및 텔레포니 컨텍스트 통합의 맥락에서 DTMF는 전화 통화 중, 특히 대화형 음성 응답(IVR) 시스템에서 신호 입력 메커니즘으로 사용됩니다. DTMF 입력을 통해 에이전트는 다음을 수행할 수 있습니다.

- 메뉴 선택 항목을 인식합니다(예: 결제는 "1을 누릅니다. 지원하려면 2를 누릅니다.")
- 숫자 입력 수집(예: 계정 번호, PINs 및 확인 번호)
- 통화 흐름에서 워크플로 또는 상태 전환 트리거

- 필요한 경우 음성에서 터치 톤으로 되돌리기

1. LLM 스트림 컨텍스트를 통한 이유

- 쿼리는 에이전트로 전송되어 세션 메타데이터(예: 호출자 ID, 이전 컨텍스트)와 함께 에이전트를 LLM으로 전달합니다.
- LLM은 아마도 chain-of-thought 전략을 사용하거나 상호 작용이 진행 중인 경우 멀티턴 메모리를 사용하여 응답을 생성합니다.

2. 음성 응답을 반환합니다.

- 에이전트는 TTS(text-to-speech 음성 변환)를 사용하여 응답을 음성으로 변환합니다.
- 음성 채널을 통해 사용자에게 오디오를 반환합니다.

기능

- 실시간 음성 이해 및 생성
- STT 및 TTS를 지원하는 다국어 I/O
- 텔레포니 또는 스트리밍 APIs와 통합
- 전환 간 세션 인식 및 메모리 핸드오프

일반 사용 사례

- 대화형 IVR 시스템
- 가상 리셉셔니스트 및 약속 스케줄러
- 음성 기반 헬프데스크 에이전트
- 웨어러블 보이스 어시스턴트
- 스마트 홈 및 접근성 도구를 위한 음성 인터페이스

구현 지침

다음 도구 및를 사용하여이 패턴을 빌드할 수 있습니다 AWS 서비스.

- STT용 Amazon Lex V2 또는 Amazon Transcribe
- TTS용 Amazon Polly

- 스트리밍 및 텔레포니를 위한 Amazon Chime SDK, Amazon Connect 고객 또는 Amazon Interactive Video Service(Amazon IVS)
- Anthropic, AI21 또는 기타 파운데이션 모델을 사용한 추론을 위한 Amazon Bedrock
- AWS Lambda STT, LLM, TTS 및 세션 컨텍스트를 연결하는 방법

(선택 사항) 추가 개선 사항에는 다음이 포함될 수 있습니다.

- 컨텍스트 인식 RAG용 Amazon Kendra 또는 OpenSearch
- 세션 메모리용 Amazon DynamoDB
- 추적성을 AWS X-Ray 위한 Amazon CloudWatch Logs 및

요약

음성 및 음성 에이전트는 자연스러운 대화를 통해 상호 작용하는 지능형 시스템입니다. 음성 인터페이스를 LLM 추론 및 실시간 스트리밍 인프라와 통합하여 음성 에이전트는 원활하고 액세스 가능하며 확장 가능한 상호 작용을 가능하게 합니다.

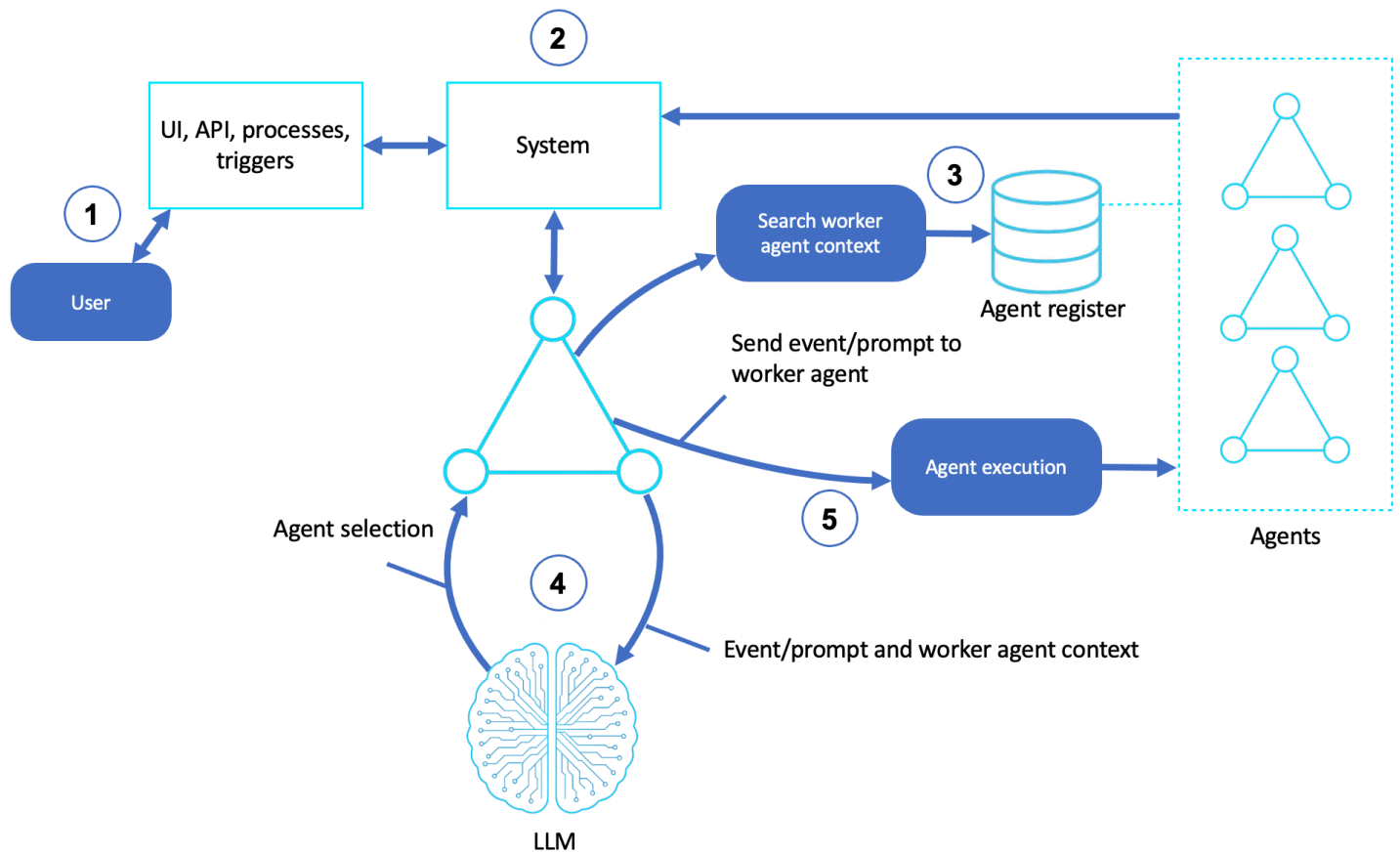
워크플로 오케스트레이션 에이전트

워크플로 오케스트레이션 에이전트는 분산 시스템에서 다단계 작업, 프로세스 및 서비스를 관리하고 조정합니다. 이러한 에이전트는 추론하고 독립적으로 행동하는 대신 하위 에이전트 또는 다른 시스템에 작업을 위임하고 실행 컨텍스트를 유지하며 중간 결과에 따라 조정합니다.

이러한 에이전트는 자동화 흐름의 기본 부분입니다. 다양한 에이전트와 도구를 순차적으로 또는 조건부로 호출해야 하는 장기 실행 작업, 다중 에이전트 구성 및 교차 도메인 통합을 처리할 때 특히 유용합니다.

아키텍처

워크플로 오케스트레이션 에이전트는 다음 다이어그램에 나와 있습니다.



설명

1. 사용자 입력을 수신합니다.

- 사용자(또는 외부 트리거)는 UI, API 또는 시스템 이벤트를 통해 작업을 시작합니다.

2. 시스템 이벤트 처리

- 시스템 구성 요소는 요청을 수신하고 오케스트레이션이 필요한 이벤트 또는 명령을 내보냅니다.

3. 컨텍스트를 검색합니다.

- 워크플로 에이전트는 지식 기반 및 에이전트 레지스트리를 쿼리하여 메타데이터, 도메인 및 이전 성공률을 기반으로 작업에 적합한 작업자 에이전트를 찾습니다.

4. LLM 에이전트를 선택합니다.

- LLM은 작업 설명과 사용 가능한 옵션을 분석하여 최상의 에이전트 또는 워크플로 계획을 선택하는 데 도움이 됩니다.
- 또한 선택한 에이전트에게 전송할 작업별 프롬프트를 공식화할 수도 있습니다.

5. 위임 및 실행

- 선택한 작업자 에이전트가 이벤트 또는 프롬프트를 수신하고 명령 실행을 시작합니다.

- 실행 상태를 추적하고, 실패 시 재시도하고, 중간 결과를 시퀀스의 다음 에이전트에 전달할 수 있습니다.

기능

- 에이전트 구성(예: 감독자, 공동 작업자 에이전트 및 도구)
- 이벤트 기반 또는 예약된 실행
- 시간 경과에 따른 메모리 및 상태 추적
- 계층적 또는 병렬 작업 오케스트레이션(비동기식 워크플로와 비동기식 워크플로 비교)
- 동적 에이전트 선택 및 연결

일반 사용 사례

- 다단계 자동화(예: 데이터 수집 및 보고)
- 고객 서비스 라우팅 및 에스컬레이션(예: agent-as-coordinator)
- AI 에이전트는 동일한 루프 내에서 인간 및 봇과 조정
- LLM 기반 로직을 사용하여 엔터프라이즈 프로세스를 자동화합니다.
- 하이브리드 시스템은 AI 에이전트와 기존 오케스트레이션 도구를 결합합니다.

구현 지침

다음 도구 및를 사용하여이 패턴을 빌드할 수 있습니다 AWS 서비스.

- 추론 및 에이전트 선택을 위한 Amazon Bedrock
- AWS Step Functions 워크플로 구성을 위한 또는 Amazon EventBridge
- AWS Lambda 실행 단위 또는 작업 실행기로 사용
- 상태 및 결과를 추적하기 위한 Amazon DynamoDB, Amazon Simple Storage Service(Amazon S3) 또는 Amazon RDS
- AWS AppFabric 교차 시스템 조정을 위한 또는 Amazon AppFlow
- (선택 사항) Amazon SageMaker 실행 에이전트를 사용하여 도메인별 작업자 에이전트 호스팅

요약

워크플로 에이전트는 다중 에이전트 환경에서 목표를 조정, 조정 및 조정합니다. 즉, AI 에이전트는 설명 가능한 모듈식 워크플로를 통해 협업하고, 런타임 조건에 적응하고, 복잡한 결과를 제공할 수 있습니다.

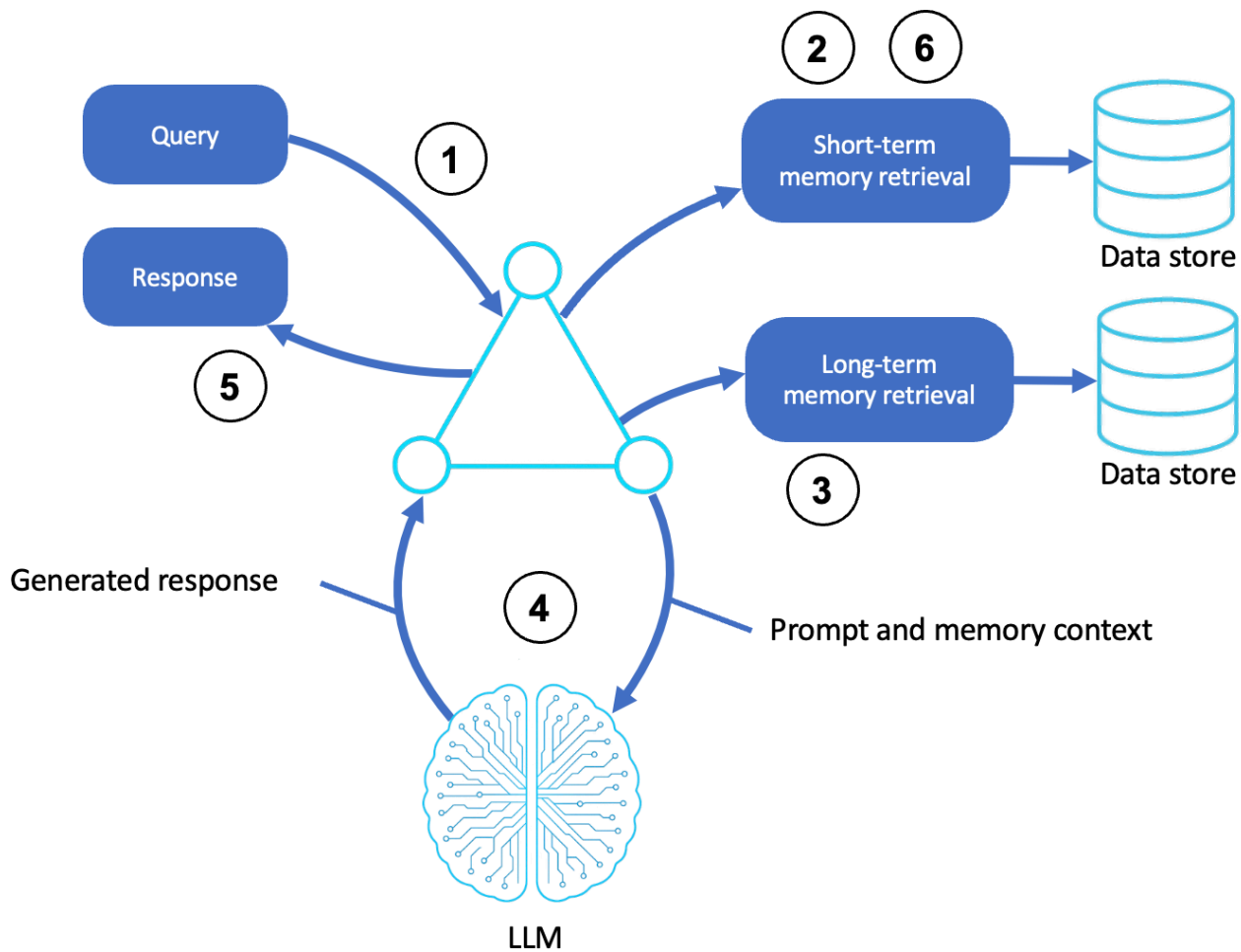
메모리 증강 에이전트

단기 및 장기 메모리를 사용하여 저장, 검색 및 추론할 수 있는 기능으로 메모리 증강 에이전트가 향상됩니다. 이를 통해 여러 작업, 세션 및 상호 작용에서 컨텍스트를 유지할 수 있으므로 보다 일관성 있고 개인화된 전략적 응답을 얻을 수 있습니다.

상태 비저장 에이전트와 달리 메모리 증강 에이전트는 과거 데이터를 참조하여 적응하고, 이전 결과에서 학습하고, 사용자의 목표, 기본 설정 및 환경에 맞는 결정을 내립니다.

아키텍처

메모리 증강 에이전트는 다음 다이어그램에 나와 있습니다.



설명

1. 입력 또는 이벤트를 수신합니다.

- 에이전트는 사용자 쿼리 또는 시스템 이벤트를 수신합니다. 텍스트, API 트리거 또는 환경 변경일 수 있습니다.

2. 단기 메모리를 검색합니다.

- 에이전트는 세션 또는 워크플로와 관련된 최근 대화 기록, 작업 컨텍스트 또는 시스템 상태를 검색합니다.

3. 장기 메모리를 검색합니다.

- 에이전트는 장기 메모리(예: 벡터 데이터베이스 및 키-값 저장소)를 쿼리하여 다음과 같은 과거 인사이트를 확인합니다.
 - 사용자 기본 설정
 - 과거 결정 및 결과

- 학습한 개념, 요약 또는 경험

4. LLM을 통한 이유

- 메모리 컨텍스트는 LLM 프롬프트에 내장되어 에이전트가 현재 입력과 이전 지식을 모두 기반으로 추론할 수 있습니다.

5. 출력을 생성합니다.

- 에이전트는 작업 기록 및 사용자의 입력에 따라 개인화된 컨텍스트 인식 응답, 계획 또는 작업을 생성합니다.

6. 메모리 업데이트

- 업데이트된 목표, 성공 및 실패 신호, 구조화된 응답과 같은 새로운 정보는 향후 작업을 위해 저장됩니다.

기능

- 대화 또는 이벤트 간 세션 연속성
- 시간 경과에 따른 목표 지속성
- 진화하는 상태에 따른 상황 인식
- 이전 성공 및 실패로 인한 적응성
- 사용자 기본 설정 및 기록과 일치하는 개인화

일반 사용 사례

- 사용자 기본 설정을 기억하는 대화식 코파일럿
- 코드베이스 변경 사항을 추적하는 코딩 에이전트
- 작업 기록에 따라 조정되는 워크플로 에이전트
- 시스템 지식에서 진화하는 디지털 트윈
- 중복 검색을 방지하는 연구 에이전트

메모리 증강 에이전트 구현

메모리 증강 에이전트에는 다음 도구 및 AWS 서비스 를 사용합니다.

메모리 계층	AWS 서비스	용도
--------	---------	----

단기	Amazon DynamoDB, Redis, Amazon Bedrock 컨텍스트	최근 상호 작용 상태의 빠른 검색
장기(정형)	Amazon Aurora, Amazon DynamoDB, Amazon Neptune	팩트, 관계 및 로그
장기(시맨틱)	OpenSearch, PostgreSQL, Pinecone	임베딩 기반 검색(즉, RAG)
스토리지	Amazon S3	트랜스크립트, 구조화된 메모리 및 파일 저장
오케스트레이션	AWS Lambda 또는 AWS Step Functions	메모리 주입 및 업데이트 수명 주기 관리
추론	Amazon Bedrock	메모리 프롬프트가 있는 Anthropic Claude 또는 Mistral

메모리 주입 프롬프트 구현

메모리를 에이전트 추론에 통합하려면 구조화된 상태와 검색 증강 컨텍스트 주입의 조합을 사용합니다.

- 언어 모델에 대한 프롬프트를 구성할 때 최신 에이전트 상태 및 최근 대화 기록을 구조화된 입력으로 포함시켜 전체 컨텍스트로 추론할 수 있도록 합니다.
- 검색 증강 생성(RAG)을 사용하여 장기 메모리에서 관련 문서 또는 사실을 가져옵니다.
- 압축 및 관련성에 대한 이전 계획, 컨텍스트 및 상호 작용을 요약합니다.
- 추론 중에 벡터 저장소 또는 구조화된 로그와 같은 외부 메모리 모듈을 주입하여 의사 결정을 안내합니다.

요약

메모리 증강 에이전트는 경험을 통해 배우고 사용자 컨텍스트를 기억하여 사고 연속성을 유지합니다. 이러한 에이전트는 장기 협업, 개인화 및 전략적 추론을 사용하여 사후 대응 인텔리전스를 능가합니다. 에이전트 AI 측면에서 메모리를 사용하면 에이전트가 적응형 디지털 대응 도구처럼 동작하고 상태 비 저장 도구처럼 동작하지 않을 수 있습니다.

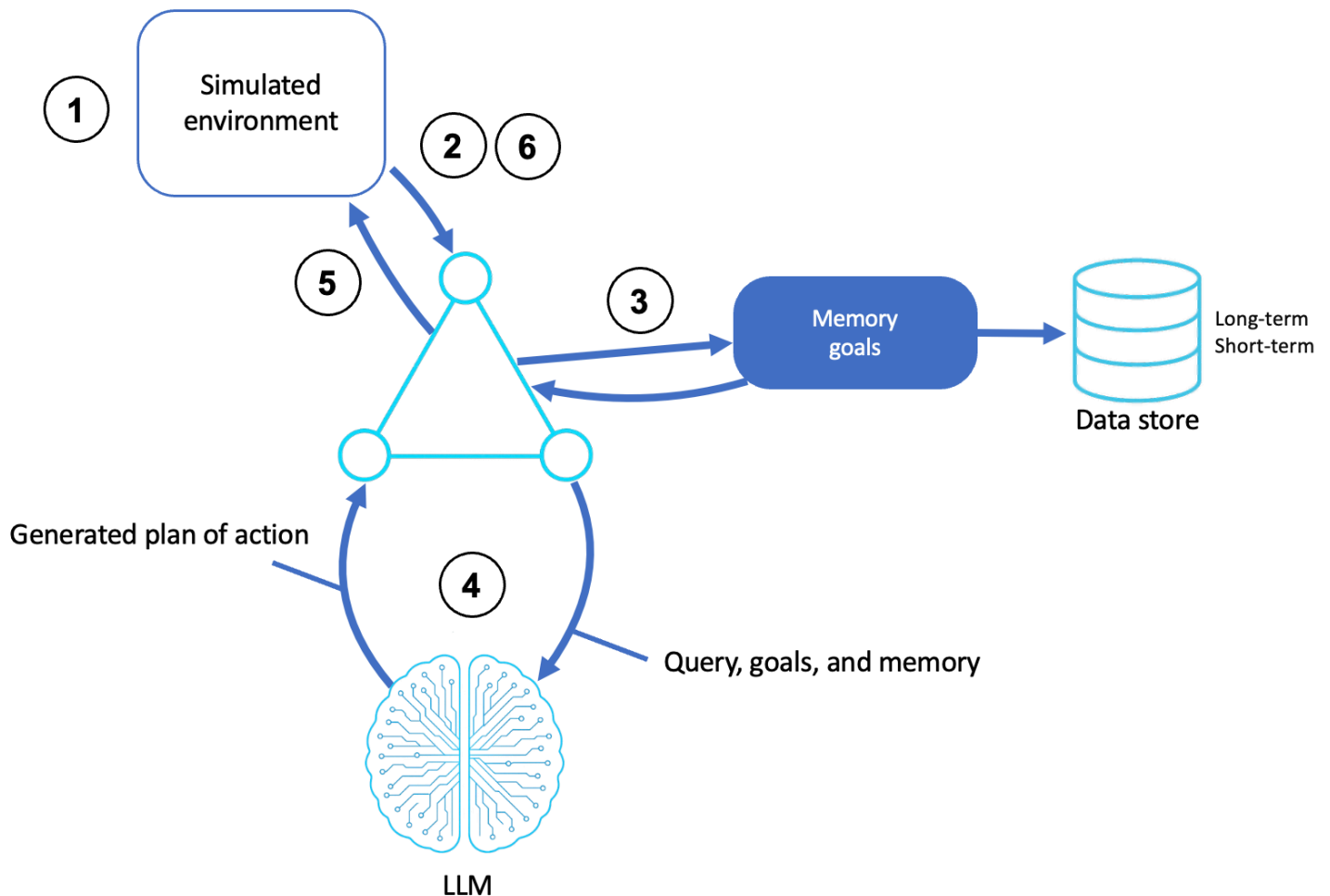
시뮬레이션 및 테스트베드 에이전트

시뮬레이션 및 테스트 기반 에이전트는 추론, 행동 및 학습을 하는 가상화되거나 제어된 환경에서 작동합니다. 이러한 에이전트는 동작을 시뮬레이션하고, 결과를 모델링하고, 전략을 실제 환경에 적용하기 전에 반복 가능한 설정으로 전략을 훈련합니다.

이 패턴은 반복 개발, 강화 학습(RL), 자율 의사 결정 평가 및 긴급 동작 테스트에 유용합니다. 시뮬레이션 에이전트는 종종 닫힌 루프에서 작동하여 환경으로부터 피드백을 받고 그에 따라 동작을 조정하므로 공간 추론, 실시간 제어 또는 복잡한 시스템 역학과 관련된 작업에 매우 중요합니다.

아키텍처

다음 다이어그램은 시뮬레이션 또는 테스트베드 에이전트를 보여줍니다.



설명

1. 환경을 시작합니다.

- 에이전트는 시뮬레이션된 환경(예: 3D 월드, 물리 엔진, CLI 샌드박스 또는 합성 데이터 스트림)을 시작합니다.
 - 에이전트는 초기 작업, 목표 또는 정책을 사용하여 환경에 로드됩니다.
2. 에이전트를 인식합니다.
- 에이전트는 시뮬레이션 원격 측정(예: 센서 에뮬레이션, 가상 카메라 및 구조화된 로그)을 통해 현재 상태를 인식합니다.
3. 목표 및 메모리를 검색합니다.
- 에이전트가 할당된 목표, 시나리오 지침 또는 컨텍스트 목표를 검색합니다.
 - 또한 다음을 포함하여 이전 메모리를 검색할 수 있습니다.
 - 장기 전략 또는 정책
 - 환경 맵 또는 알려진 제약 조건
 - 유사한 시뮬레이션의 과거 성공 또는 실패
4. 이유 및 계획
- LLM은 시뮬레이션된 상태, 작업 목표 및 학습된 지식을 해석합니다.
 - 작업 계획 또는 제어 명령을 생성합니다.
5. 시뮬레이션된 작업을 실행합니다.
- 에이전트는 계획을 실행하거나, 상태를 수정하거나, 공간을 탐색하거나, 가상 엔터티와 상호 작용합니다.
6. 학습
- 에이전트가 작업 결과 평가
 - 에이전트의 구성에 따라 다음을 수행할 수 있습니다.
 - RL 수행
 - 향후 미세 조정을 위한 로그 결과
 - 실시간 전략 조정

기능

- 합성 또는 가상 환경에서 작동
- trial-and-error 학습, 정책 세분화 및 시스템 모델링 지원
- 동작, 장애 처리 및 엣지 케이스에 대한 위험이 낮은 테스트
- 다중 에이전트 설정에서 긴급 에이전트 동작 분석 활성화

- 폐쇄 루프 제어 및 human-in-the-loop 루프 탐색 모두 지원

일반 사용 사례

- 로봇, 드론 및 게임을 위한 강화 학습
- 가상 도로에서의 자율 차량 훈련
- DevOps 및 테스트베드 시나리오를 위한 시뮬레이션된 UIs 또는 CLIs
- 소셜 시뮬레이션에서 발생한 행동 실험
- 프로덕션 전 결정 로직의 안전 검증

구현 지침

다음 도구 및를 사용하여 시뮬레이션 및 테스트용 에이전트를 빌드할 수 있습니다 AWS 서비스.

구성 요소	AWS 서비스	용도
환경	Amazon SageMaker 스튜디오 랩의 Amazon ECS, Amazon Amazon EC2 또는 사용자 지정 시뮬레이터	가상 세계(Gazebo, Unity, Unreal) 또는 샌드박스 CLIs 실행
에이전트 로직	Amazon Bedrock, Amazon SageMaker 또는 AWS Lambda	LLM 기반 플래너 또는 RL 에이 전트
피드백 루프	Amazon SageMaker 강화 학 습, Amazon CloudWatch 또는 사용자 지정 로그	보상 추적, 결과 점수 및 동작 로깅
메모리 및 재생	Amazon S3, Amazon DynamoDB 또는 Amazon RDS	영구 상태, 에피소드 기록 또는 시나리오 데이터
시각화	Amazon CloudWatch 대시보드 또는 Amazon SageMaker 노트 북	정책 변경 사항, 결과 및 훈련 지표 관찰

다음은 추가 애플리케이션입니다.

- [AWS SimSpace Weaver](#) 대규모 공간 시뮬레이션용
- [AWS IoT Core](#) 새도우 디바이스 테스트용
- 에이전트 평가 및 벤치마킹을 위한 [Amazon SageMaker 실험](#)

요약

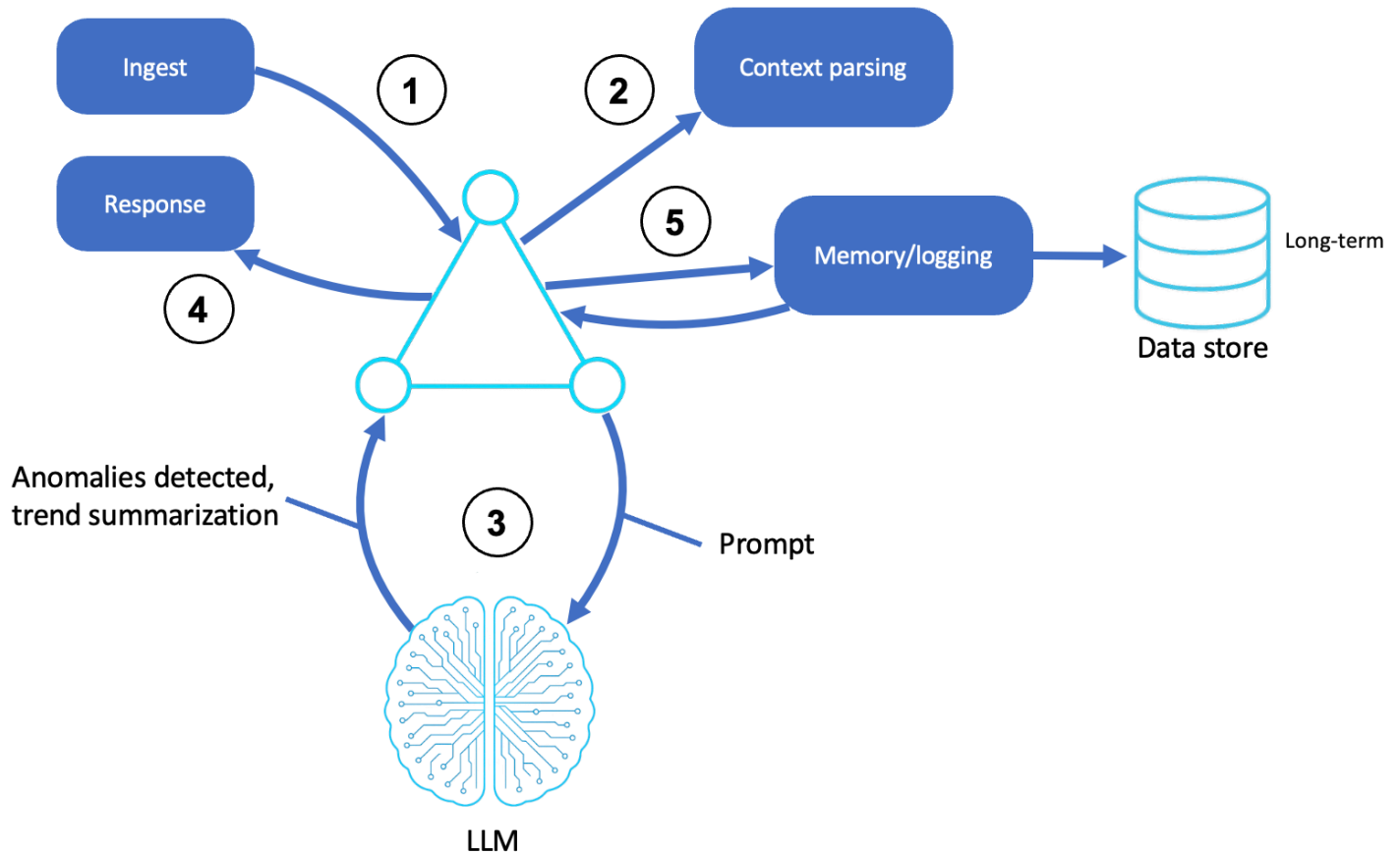
시뮬레이션 및 테스트용 에이전트는 프로덕션 시스템에 배포되기 전에 구조화된 탐색을 위한 것입니다. 이러한 에이전트를 사용하여 자율 탐색 정책을 훈련하고, 합성 환경에서 비즈니스 프로세스를 테스트하고, 조정 패턴에 대한 스웜을 평가할 수 있습니다.

관찰자 및 모니터링 에이전트

관찰자 및 모니터링 에이전트는 시스템, 환경 및 상호 작용을 수동적으로 관찰하여 패턴을 감지하고 인사이트를 생성하며 작업을 트리거합니다. 지능형 감시자는 동작을 직접 시작하지 않고도 알림, 진단 및 감사를 개선합니다.

이러한 에이전트는 특히 AI-in-the-loop 모니터링, 이상 탐지, 규정 준수 감독 및 보안 인텔리전스에서 기존 모니터링에 적응성 또는 추론이 부족한 경우에 뛰어납니다. 옵저버 에이전트는 시스템 원격 측정 및 사용자 상호 작용을 지속적으로 모니터링하는 이벤트 리스너입니다. 에이전트는 지각, 해석 및 조건부 에스컬레이션 또는 보고에 따라 달라집니다.

아키텍처



설명

1. 원격 측정 수집

- 에이전트는 다음과 같은 하나 이상의 시스템 소스로부터 입력을 받습니다.
 - 로그(애플리케이션, 인프라, 보안)
 - 지표(성능, 지연 시간, 사용량)
 - 이벤트(API 호출, 사용자 작업, 센서 데이터)

2. 구문 분석 컨텍스트

- 원시 입력은 구문 분석되고 구조화되며 타임스탬프, 액터 자격 증명, 시스템 상태 및 추적 ID와 같은 메타데이터로 보강됩니다.

3. LLM을 사용하는 이유

- 에이전트는 LLM 또는 로직 모듈을 사용하여 이상을 식별하고, 추세를 요약하고, 분산 트레이스 또는 기간 간에 상호 연관시켜 구문 분석된 입력을 해석합니다.

4. 분류 또는 알림

- 에이전트는 관찰된 동작이 다음을 정당화하는지 확인합니다.
 - 알림 또는 에스컬레이션
 - 보고서 또는 대시보드 업데이트
 - 응답 트리거(예: 자동 문제 해결 및 정책 적용)

5. 로그 메모리 또는 피드백 루프

- 에이전트는 장기 학습, 감사 또는 다른 에이전트에 대한 향후 참조를 위해 이벤트와 결정을 저장합니다.

기능

- 수동 및 비침습적(에이전트가 직접 작동하지 않음)
- 확장성과 비동기성이 뛰어남
- 노이즈 또는 분산 신호 간 AI 기반 상관관계
- 감사, 규정 준수 및 실시간 인사이트 지원
- 다운스트림 에이전트 또는 인적 워크플로에 피드 가능

일반 사용 사례

- 마이크로서비스 및 APIs에 대한 AI 증강 관찰성
- 모델 드리프트, 정책 위반 또는 out-of-band 동작 모니터링
- 고객 활동 분석 또는 상호 작용 요약
- 커밋 또는 배포를 모니터링하는 코드 검토 에이전트
- LLM 추론을 사용한 보안 또는 규정 준수 로그 모니터링

구현 지침

다음 도구 및를 사용하여 관찰자 및 모니터링 에이전트를 빌드할 수 있습니다 AWS 서비스.

구성 요소	AWS 서비스	용도
이벤트 수집	Amazon EventBridge, Amazon CloudWatch Logs, Amazon Kinesis, Amazon S3	정형 및 비정형 원격 측정 수집

사전 처리	AWS Lambda, AWS Glue, AWS Step Functions	원시 데이터를 구조화된 흐름 포트로 변환
추론 엔진	Amazon Bedrock, Amazon SageMaker, AWS Lambda	이벤트 분석, 동작 분류, 인사이 트 생성
스토리지 및 메모리	Amazon S3, Amazon DynamoDB, OpenSearch	지속적인 관찰, 요약 및 출력
알림 및 에스컬레이션	Amazon SNS, AWS AppFabric , Amazon EventBridge	다운스트림 시스템 또는 에이 전트 트리거

다음은 추가 애플리케이션입니다.

- [AWS Security Hub CSPM](#) 보안 로그 모니터링을 위한
- 에이전트 출력을 시각화하기 위한 [Amazon Quick](#)

요약

관찰자 및 모니터링 에이전트는 시스템과 동작을 실시간으로 추적합니다. 인간 또는 규칙이 간과할 수 있는 패턴을 식별하여 이상을 탐지하고, 보안을 감사하고, 운영 인텔리전스를 수집합니다. 이 기능은 변화하는 조건에 적응하고 포괄적인 데이터 분석을 기반으로 결정을 내릴 수 있는 시스템을 만드는 데 도움이 됩니다.

다중 에이전트 공동 작업

다중 에이전트 공동 작업은 각각 고유한 역할, 전문화 또는 목표를 가진 여러 자율 에이전트가 협상하여 복잡한 작업을 해결하는 패턴을 말합니다. 이러한 에이전트는 정보를 공유하고, 책임을 나누고, 목표에 대한 공동 추론을 통해 독립적으로 또는 다른 에이전트와 함께 운영할 수 있습니다.

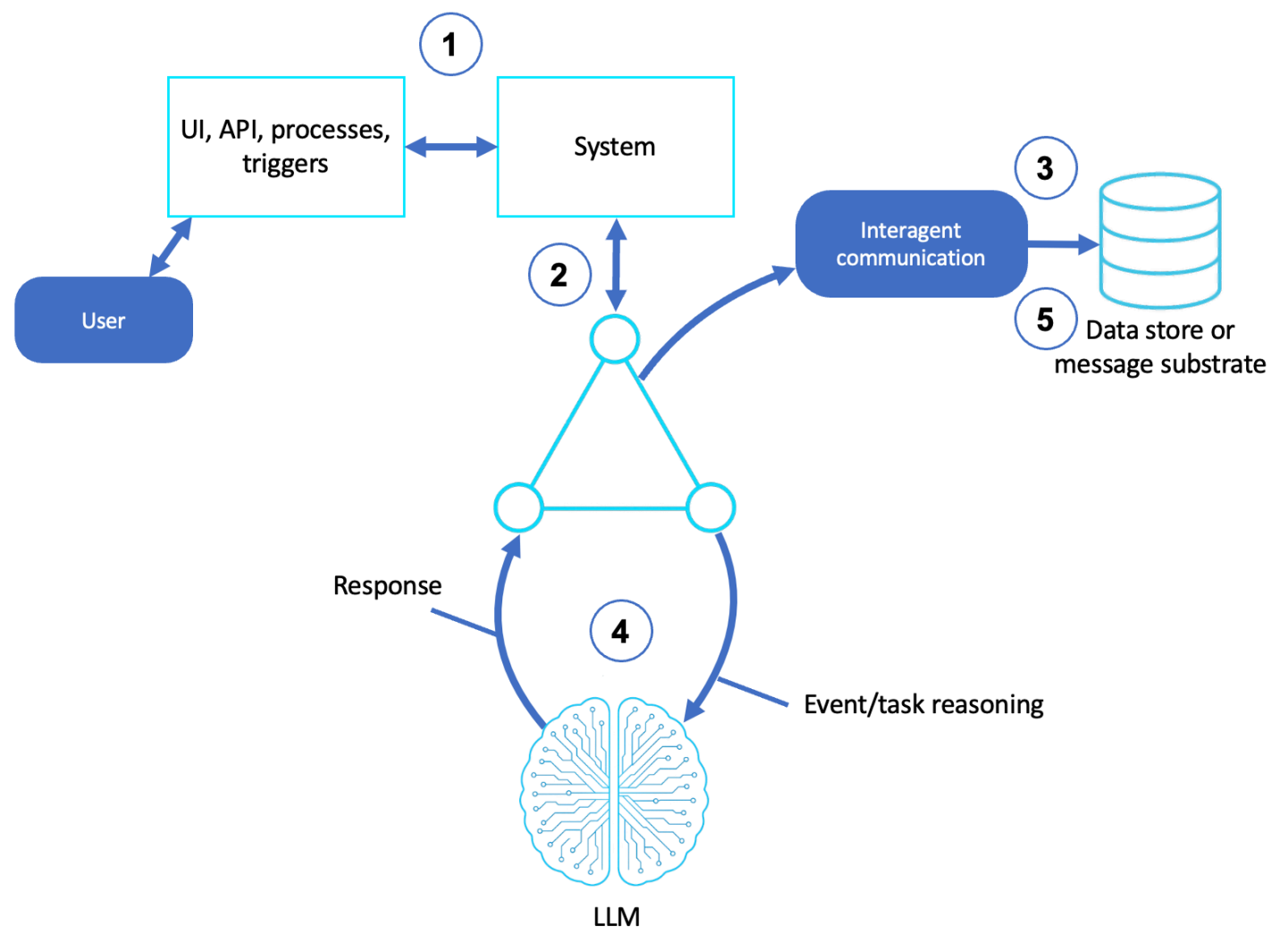
이 패턴은 구조화된 흐름에서 하위 에이전트에게 작업을 중앙에서 조정하고 위임하는 워크플로 에이전트와 다릅니다. 반대로 다중 에이전트 공동 작업은 적응성, 병렬 처리 및 인식 분할을 활성화하여 peer-to-peer 또는 긴급 조정을 강조합니다. 다음 표에서는 다중 에이전트 공동 작업을 워크플로 에이전트와 비교합니다.

Feature	워크플로 에이전트	용도
---------	-----------	----

컨트롤	중앙 집중식 코디네이터	분산형, 분산형 또는 역할 기반 피어
상호 작용	에이전트 한 명이 실행을 위임하고 추적합니다.	여러 에이전트가 협상, 공유 및 조정
설계	미리 정의된 작업 시퀀스	긴급하고 유연한 작업 배포
조정	절차 오케스트레이션	협력적 또는 경쟁적 상호 작용
사용 사례	엔터프라이즈 프로세스 자동화	복잡한 추론, 탐색 및 긴급 전략

아키텍처

다음 다이어그램은 다중 에이전트 공동 작업을 보여줍니다.



설명

1. 작업을 시작합니다.
 - 사용자 또는 시스템이 높은 수준의 목표 또는 문제를 내보냅니다.
 - "관리자" 에이전트 또는 시작 컨텍스트가 목표를 정의합니다.
2. 역할을 할당하거나 검색합니다.
 - 에이전트가 플래너, 연구원, 실행자, 비평가 또는 설명자와 같은 다른 역할에 자체 할당(대칭 로직 또는 추론)되거나 위임(이벤트 브로커)됩니다.
3. 다른 에이전트와 통신합니다.
 - 에이전트는 공유 메모리, 메시징 대기열 또는 프롬프트 체인을 통해 통신합니다.
 - 서로 토론하거나 쿼리하거나 하위 작업을 제안할 수 있습니다.
4. 특수 추론 사용
 - 각 에이전트는 자체 모델 또는 도메인 로직을 사용하여 문제의 부분을 해결합니다.
 - 에이전트는 역할별 프롬프트 및 메모리와 함께 LLMs를 사용할 수 있습니다.
5. 출력 또는 목표를 조정합니다.
 - 에이전트는 기여를 최종 답변, 계획 또는 작업으로 합성합니다.
 - (선택 사항) 감독 에이전트는 합성된 출력을 검증하거나 요약할 수 있습니다.

기능

- 전문 역할 또는 기술을 갖춘 피어 레벨 에이전트
- 커뮤니케이션 또는 협상을 통해 발생한 행동
- 복잡하거나 다면적인 문제의 병렬 처리
- 속고, 자체 수정 및 반사 반복 지원
- 소셜 역학, 과학 협업 또는 엔터프라이즈 팀 역할 모델링

일반 사용 사례

- 자율 연구 팀(검색 에이전트, 요약자 및 검사기)
- 소프트웨어 개발(플래너, 코더 및 테스터)
- 비즈니스 시나리오 모델링(재무, 정책 및 규정 준수)
- 협상, 입찰 또는 다자간 추론

- 멀티모달 작업(이미지, 텍스트 및 로직)

구현 지침

다음 도구 및를 사용하여 다중 에이전트 시스템을 빌드할 수 있습니다 AWS 서비스.

구성 요소	AWS 서비스	용도
에이전트 호스팅	Amazon Bedrock, Amazon SageMaker, AWS Lambda	개별 LLM 기반 에이전트 호스팅
통신 계층	Amazon SQS, Amazon EventBridge, AWS AppFabric	에이전트 간의 메시징 및 조정
공유 메모리	Amazon DynamoDB, Amazon S3 또는 OpenSearch	다중 에이전트 메모리 또는 블랙보드 시스템
오케스트레이션 계층	AWS Step Functions, AWS Lambda 파이프라인	킥오프, 제한 시간, 대체 및 재시도 로직
에이전트 식별	Amazon Bedrock 에이전트(역할 정의) AWS AppConfig 및 Amazon Bedrock Converse API(Amazon Bedrock 외부의 에이전트)	역할 기반 도구 또는 에이전트 호출 및 경계 적용
발생한 상호 작용	Amazon EventBridge 파이프라인 또는 에이전트 레지스트리	동적 작업 라우팅 또는 에스컬레이션 활성화

요약

다중 에이전트 공동 작업은 모듈식 역할 기반 에이전트에 문제 해결 작업을 분산합니다. 워크플로 오케스트레이션과 달리 협업 패턴은 인간이 문제를 해결하는 방법을 반영하는 새로운 인텔리전스, 복원력 및 확장성을 사용합니다. 개방형 도메인, 창의적인 작업, 멀티모달 추론 및 다양한 관점에서 이점을 얻을 수 있는 환경에 특히 유용합니다.

결론

앞서 설명한 패턴은 에이전트 AI의 실제 구현에 대한 기본 접근 방식을 보여줍니다. 기본 추론에서 메모리 증강 인텔리전스에 이르기까지 각 패턴은 자율성, 비동기성 및 기관을 기반으로 하는 지각, 인식 및 작업에 맞게 고유하게 구성됩니다.

이러한 패턴은 지능형 목표 지향 시스템을 구축하기 위한 어휘와 기술 청사진을 공유합니다. 패턴이 사용자 인터페이스에 포함되든, 클라우드 서비스를 통해 오케스트레이션되든, 에이전트 팀 간에 조정되든, 각 패턴은 조정 가능하고 모듈화됩니다.

요점

- 에이전트 패턴 구성 가능 - 대부분의 실제 에이전트는 둘 이상의 패턴(예: 도구 기반 추론 및 메모리가 있는 음성 에이전트)을 혼합합니다.
- 에이전트 설계는 상황에 따라 다름 - 상호 작용 표면, 작업 복잡성, 지연 시간 허용 범위 및 도메인별 제약 조건을 기반으로 패턴을 선택합니다.
- AWS 네이티브 구현 달성 가능 - Amazon Bedrock, Amazon SageMaker AWS Lambda AWS Step Functions 및 이벤트 기반 아키텍처를 사용하면 모든 에이전트 패턴을 대규모로 제공할 수 있습니다.

LLM 워크플로

에이전트 패턴에서는 일반적인 AI 에이전트 패턴을 살펴보았습니다. 각 패턴은 지각, 행동, 학습 및 인식과 같은 모듈식 기능 세트를 중심으로 구축되었습니다. 많은 에이전트 패턴에서 인지 모듈의 핵심은 추론, 계획 및 의사 결정을 할 수 있는 대규모 언어 모델(LLM)입니다. 그러나 LLM만 호출하는 것만으로는 지능적이고 목표 지향적인 동작을 생성하기에 충분하지 않습니다.

복잡한 작업을 안정적으로 수행하려면 에이전트가 구조화된 워크플로 내에 LLM을 포함해야 합니다. 여기서 모델의 기능은 도구, 메모리, 계획 루프 및 조정 로직으로 보강됩니다. 이러한 LLM 워크플로를 통해 에이전트는 목표를 세분화하고, 하위 작업을 라우팅하고, 외부 서비스를 호출하고, 결과를 반영하고, 다른 에이전트와 조정할 수 있습니다.

이 장에서는 재사용 가능한 워크플로를 중심으로 구성된 강력하고 확장 가능하며 지능적인 LLM 기반 인지 모듈을 구축하기 위한 핵심 설계 패턴을 소개합니다.

이 섹션의 내용

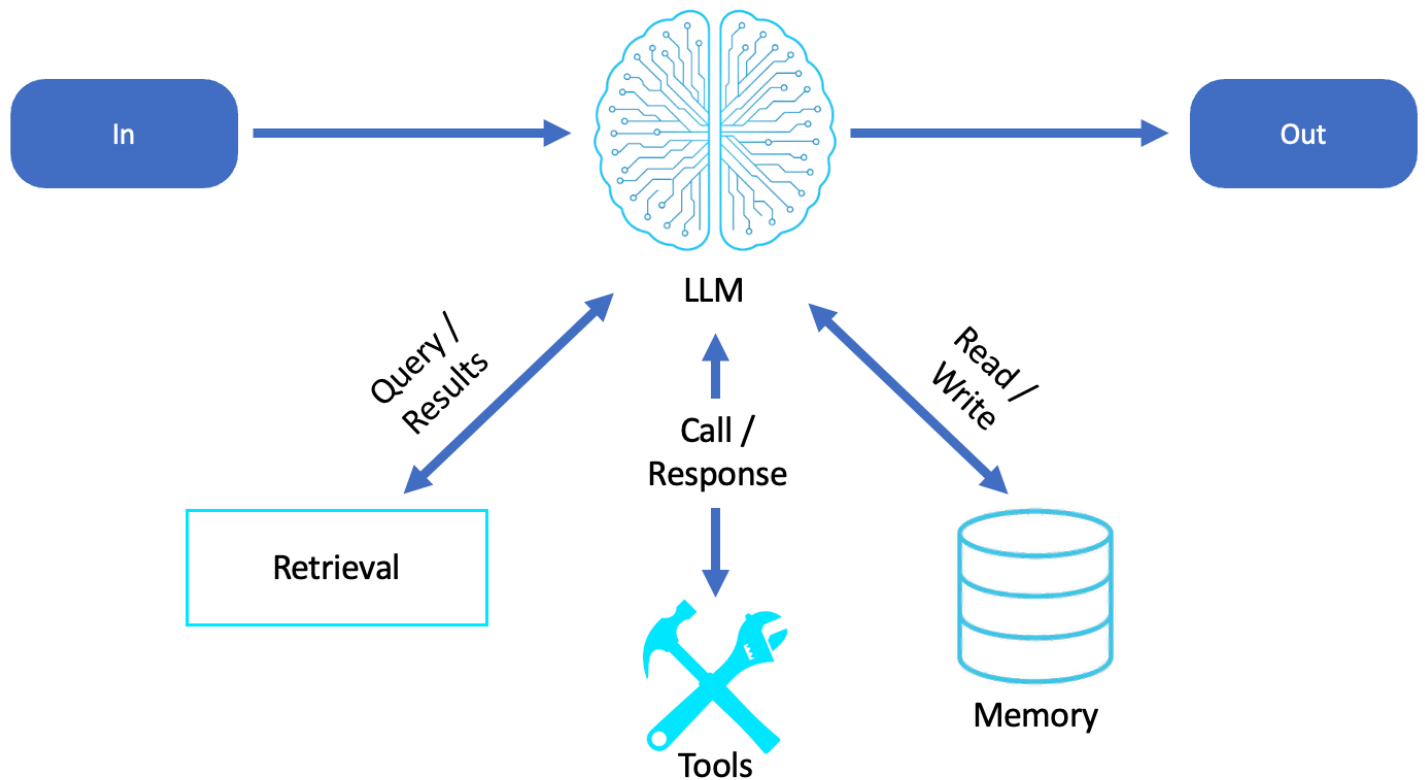
- [LLM 증강 인식 개요](#)
- [프롬프트 체인을 위한 워크플로](#)
- [라우팅을 위한 워크플로](#)
- [병렬화를 위한 워크플로](#)
- [오케스트레이션을 위한 워크플로](#)
- [평가자 및 반사 구체화 루프를 위한 워크플로](#)
- [결론](#)

LLM 증강 인식 개요

핵심에서 소프트웨어 에이전트의 인지 모듈은 증강으로 래핑된 LLM으로 볼 수 있습니다. 에이전트는 다음 구성 요소를 사용하여 환경 내에서 효과적으로 추론할 수 있습니다.

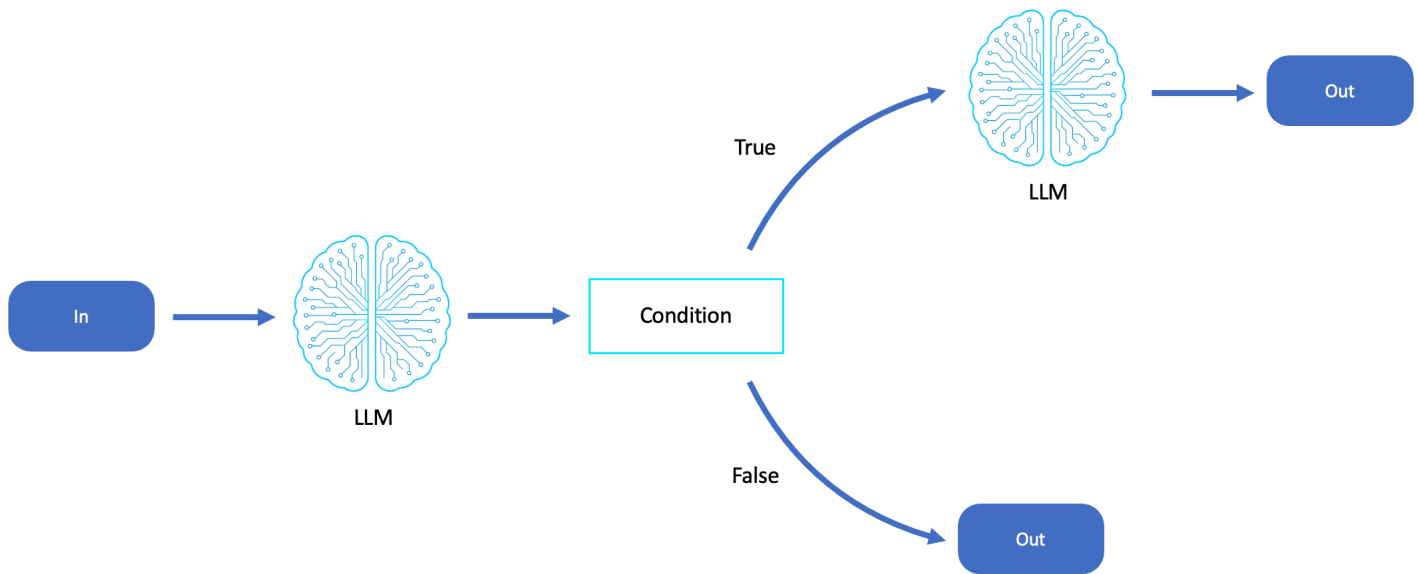
- 프롬프트 - 컨텍스트, 지침, 예제 및 메모리를 사용하여 입력 프레이밍
- 검색 - 검색 증강 생성(RAG)과 같은 벡터 검색 또는 의미 체계 메모리를 통해 LLM 프롬프트에 up-to-date 또는 도메인별 지식 제공
- 도구 사용 - LLM이 APIs를 호출하거나 함수를 호출하여 정보를 검색하거나 작업할 수 있습니다.
- 메모리 - 구조화된 데이터베이스 또는 컨텍스트 요약물 사용하여 영구 또는 세션 기반 상태를 추론 루프에 통합

이러한 증강은 시간 경과에 따라 여러 작업에서 LLM을 사용하는 방법을 정의하는 워크플로로 구성되어 상태 비저장 엔진에서 동적 추론 에이전트로 변환합니다.



프롬프트 체인을 위한 워크플로

프롬프트 체인은 복잡한 작업을 일련의 단계로 분해합니다. 여기서 각 단계는 이전 단계의 출력을 처리하거나 기반으로 빌드하는 개별 LLM 호출입니다.



프롬프트 체인 워크플로는 작업을 논리적으로 순차적 추론 단계로 나눌 수 있고 중간 출력이 다음 단계를 알려주는 시나리오에 적합합니다. 문서 검토, 코드 생성, 지식 추출 및 콘텐츠 구체화와 같은 구조화된 사고, 점진적 변환 또는 계층화된 분석이 필요한 워크플로에 탁월합니다.

설명

- 작업의 복잡성이 단일 LLM 호출의 컨텍스트 기간 또는 추론 깊이를 초과합니다.
- 한 단계의 출력(예: 분석, 요약 또는 계획)은 후속 결정 또는 생성 단계의 입력이 됩니다.
- 추론 단계에서 투명성과 제어가 필요합니다(예: 감사 가능한 중간 결과).
- 단계 사이에 외부 검증, 필터링 또는 보강 로직을 연결하려고 합니다.
- 연구 에이전트, 에디토리얼 어시스턴트, 계획 시스템, 다단계 코파일럿 등 파이프라인 스타일의 추론 루프에서 작업하는 에이전트에게 적합합니다.

기능

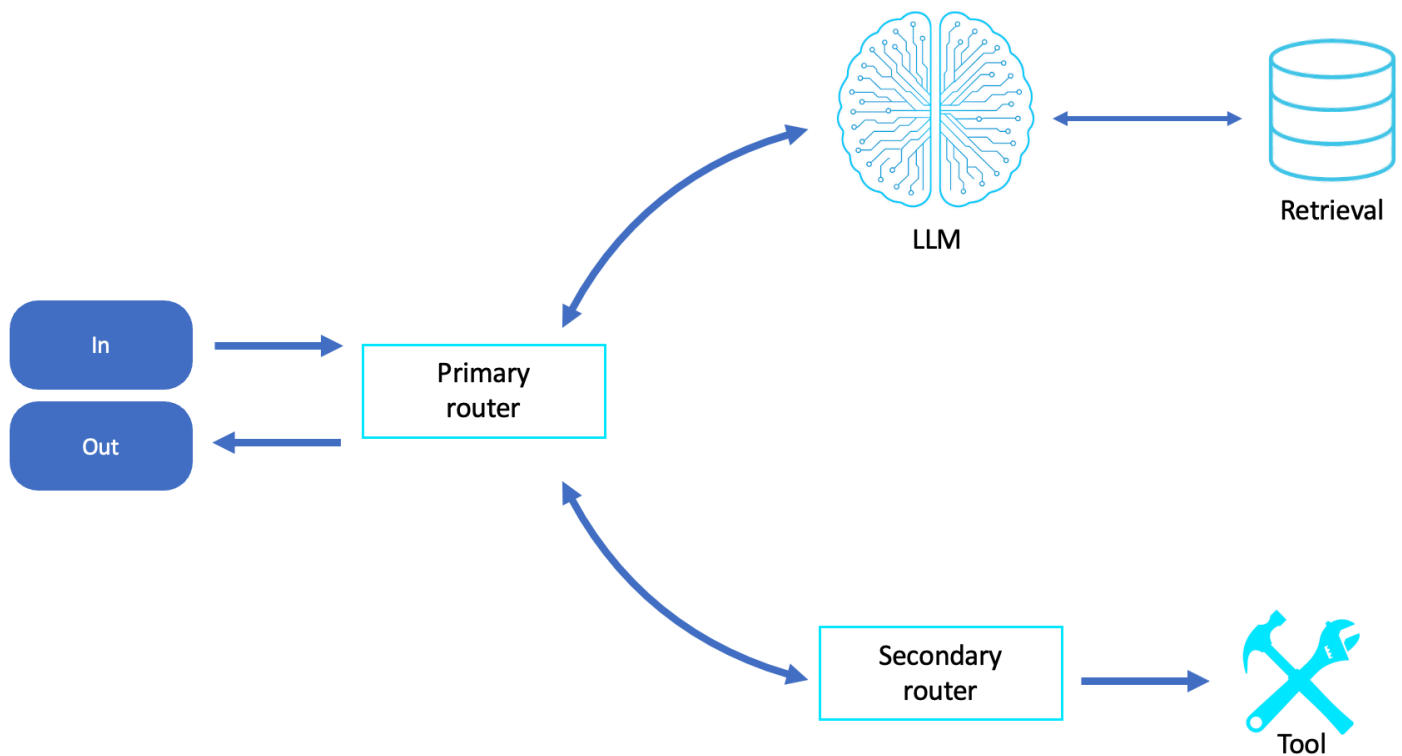
- LLM 호출의 선형 또는 분기 체인
- 구조화된 입력으로 전달되거나 후속 프롬프트에 포함된 중간 결과
- AWS Step Functions AWS Lambda 또는 에이전트별 실행기로 오케스트레이션할 수 있습니다.

일반 사용 사례

- 다단계 추론 작업(예: "평가 재작성 요약")
- 계층화된 출력을 합성하는 연구 도우미(예: "사실 답변 질문 검색")
- 코드 생성 파이프라인("계획 작성 코드 테스트 코드 설명 출력 생성")

라우팅을 위한 워크플로

라우팅 패턴에서 분류자 또는 라우터 에이전트는 LLM을 사용하여 쿼리의 의도 또는 범주를 해석한 다음 입력을 특수 다운스트림 작업 또는 에이전트로 라우팅합니다.



라우팅 워크플로는 에이전트가 입력 의도, 작업 유형 또는 도메인을 빠르게 분류한 다음 요청을 특수 하위 에이전트, 도구 또는 워크플로에 위임해야 하는 시나리오에서 사용됩니다. 이는 일반 어시스턴트 역할을 하는 에이전트, 엔터프라이즈 함수의 정문 또는 도메인에 걸쳐 있는 사용자 대면 AI 인터페이스와 같은 기능 에이전트에 특히 유용합니다.

라우팅은 다음과 같은 경우에 특히 효과적입니다.

- 다양한 작업(예: 검색, 요약, 예약, 계산)에서 요청을 분류합니다.
- 보다 전문화된 워크플로에 들어가기 전에 입력을 사전 처리하거나 정규화해야 합니다.

- 다양한 입력 유형(예: 이미지와 텍스트, 정형 쿼리와 비정형 쿼리)에는 사용자 지정 처리가 필요합니다.
- 에이전트는 대화형 전환판 역할을 하여 특수 에이전트 또는 마이크로서비스에 작업을 위임합니다.
- 이 워크플로는 도메인별 코파일럿, 고객 지원 봇, 엔터프라이즈 서비스 라우터 및 멀티모달 에이전트에서 일반적이며, 지능형 디스패치가 에이전트 동작의 품질과 효율성을 모두 결정합니다.

기능

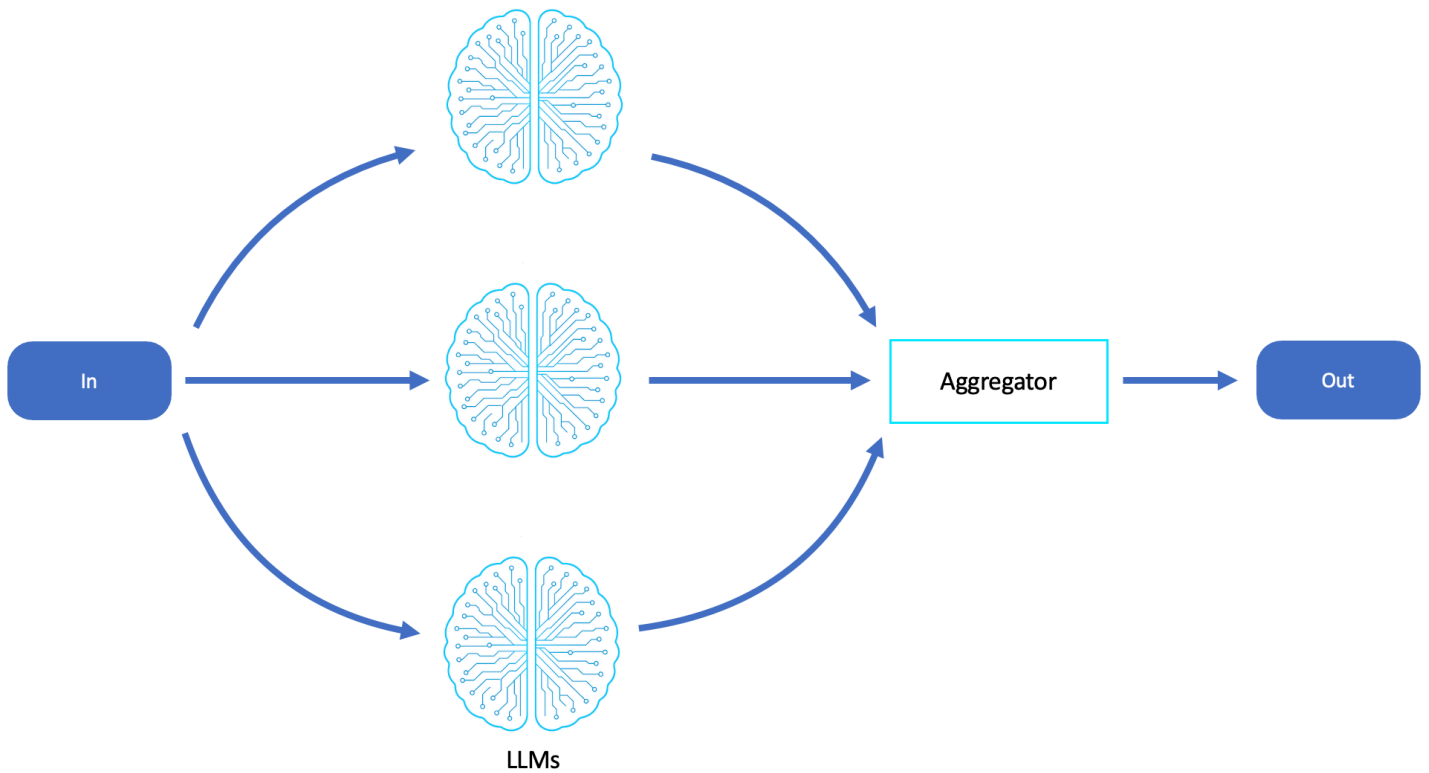
- 1차 LLM은 디스패처 역할을 합니다.
- 경로는 고유한 워크플로 또는 다른 에이전트 패턴을 호출할 수 있습니다.
- 기능의 모듈식 확장 지원

일반 사용 사례

- 다중 도메인 어시스턴트("법적, 의학적 또는 재정적 질문입니까?")
- LLM 추론으로 향상된 의사 결정 트리
- 동적 도구 선택(예: 검색 대 코드 생성)

병렬화를 위한 워크플로

이 워크플로에는 여러 LLM 호출 또는 에이전트가 동시에 처리할 수 있는 독립적인 하위 작업으로 작업을 분류하는 작업이 포함됩니다. 그런 다음 출력이 프로그래밍 방식으로 집계되고 결과로 합성됩니다.



병렬화 워크플로는 작업을 동시에 처리할 수 있는 독립적인 비순차 하위 작업으로 분할하여 효율성, 처리량 및 확장성을 크게 개선할 수 있는 경우에 사용됩니다. 에이전트가 여러 입력에 걸쳐 콘텐츠를 분석하거나 생성해야 하는 데이터가 많거나 배치 지향적이거나 다중 관점 문제 공간에서 특히 강력합니다.

병렬화는 다음과 같은 경우에 특히 효과적입니다.

- 하위 작업은 서로의 중간 결과에 의존하지 않으므로 조정 없이 병렬로 실행할 수 있습니다.
- 작업에는 여러 항목에서 동일한 추론 프로세스를 반복하는 작업이 포함됩니다(예: 여러 문서 요약 또는 옵션 목록 평가).
- 다양성, 창의성 또는 견고성을 높이기 위해 여러 가설 또는 관점을 동시에 탐색합니다.
- 동시 LLM 실행을 통해 대용량 또는 고주파 요청의 지연 시간을 줄여야 합니다.
- 이 워크플로는 문서 처리 에이전트, 설문 조사 또는 비교 엔진, 배치 요약기, 다중 에이전트 브레인스토밍, 확장 가능한 분류 또는 레이블 지정 작업, 특히 신속한 병렬 추론이 성능 이점인 경우에 일반적으로 사용됩니다.

기능

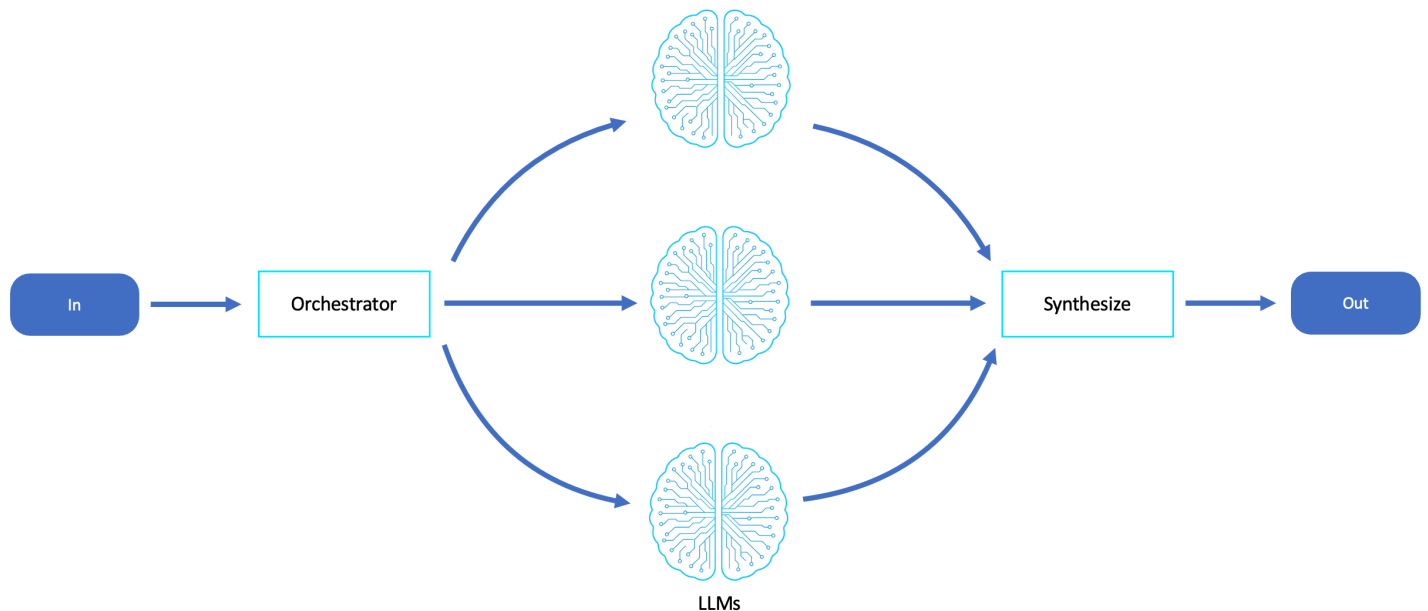
- LLM 작업의 병렬 실행(AWS Lambda AWS Fargate 또는 AWS Step Functions 맵 상태 사용)
- 합성 단계에서 결과 정렬, 검증 또는 중복 제거 필요
- 상태 비저장 에이전트 루프에 적합

일반 사용 사례

- 여러 문서 또는 관점을 병렬로 분석
- 다양한 초안, 요약 또는 계획 생성
- 배치 작업 간 처리량 가속화

오케스트레이션을 위한 워크플로

중앙 오케스트레이터 에이전트는 LLM을 사용하여 각각 특정 역할 또는 도메인 전문 지식을 갖춘 전문 작업자 에이전트 또는 모델에 하위 작업을 계획, 분해 및 위임합니다. 이는 인적 팀 구조를 미러링하고 여러 에이전트의 긴급 동작을 지원합니다.



오케스트레이션 워크플로는 구조화된 분해 및 특수 실행이 필요한 복잡하거나 계층적 또는 다학제적인 시나리오에 적합합니다. 특히 작업의 다양한 하위 구성 요소를 고유한 기능, 지식 또는 도구 세트를 가진 에이전트가 처리하는 데 가장 적합한 작업 분할이 필요한 작업에 적합합니다.

이 워크플로는 다음과 같은 경우에 특히 효과적입니다.

- 작업은 범위, 유형 또는 추론(예: 계획, 연구, 구현 및 테스트)이 다양한 하위 작업으로 나눌 수 있습니다.
- LLM 또는 메타 에이전트는 다른 에이전트를 조정하고, 진행 상황을 모니터링하고, 결과를 합성해야 합니다.
- 에이전트 책임을 모듈화하여 확장성, 재사용 및 특수 조정을 지원하려고 합니다.
- 이 시스템에는 인적 팀(예: 프로젝트 관리자, 개발자 및 검토자)이 공동으로 작동하는 방식을 모방한 역할 기반 동작이 필요합니다.

오케스트레이션은 멀티턴 계획 에이전트, 소프트웨어 개발 코파일럿, 엔터프라이즈 프로세스 에이전트 및 자율 프로젝트 실행기에 적합합니다. 중앙 집중식 작업 분석과 분산 실행 로직이 필요한 다중 에이전트 시스템을 구현할 때 특히 유용하여 에이전트 계층 간에 확장성과 설명 가능한 동작을 지원합니다.

기능

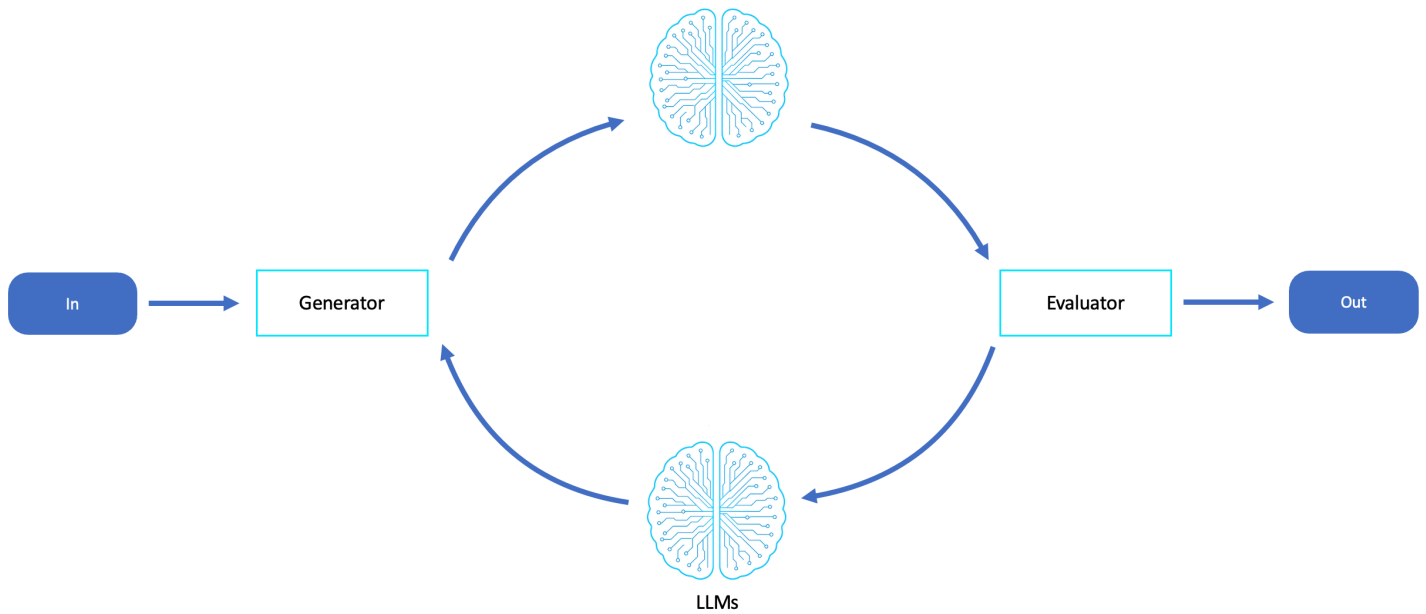
- 오케스트레이터가 목표 메타 이유 수행
- 작업자 에이전트에는 도구 액세스, 메모리 또는 도메인별 프롬프트가 포함될 수 있습니다.
- 계층적일 수 있음(다단계 작업 위임)

일반 사용 사례

- 프로젝트 관리자, 조정 연구원, 작성자 및 품질 보증 에이전트
- 계획, 실행 및 테스트를 결합하는 코파일럿 코딩
- 도구 체인 또는 API 액세스 패턴을 감독하는 에이전트

평가자 및 반사 구체화 루프를 위한 워크플로

이 워크플로는 한 LLM이 결과를 생성하고 다른 LLM이 결과를 평가하거나 평가하는 피드백 루프를 제공합니다. 이를 통해 자기 성찰, 최적화 및 반복적인 개선이 촉진됩니다.



평가자 워크플로는 출력 품질, 정확도 및 정렬이 중요하고 단일 패스 생성을 신뢰할 수 없거나 충분하지 않은 시나리오에 적합합니다. 이 워크플로는 에이전트가 더 높은 정확성 표준을 충족하거나 피드백을 기반으로 개선된 대안을 탐색하기 위해 자체 비판, 반복 및 출력을 구체화해야 하는 경우에 유용합니다.

이 워크플로는 다음과 같은 경우에 특히 효과적입니다.

- 출력에는 주관적 품질 지표(예: 스타일, 어조 및 가독성) 또는 목표 기준(예: 정확성, 안전성 및 성능)이 포함됩니다.
- 에이전트는 절충을 통해 추론하거나, 제약 조건을 평가하거나, 목표를 향해 최적화해야 합니다.
- 특히 규제, 고객 대면 또는 크리에이티브 도메인에서 기본 제공되는 중복성 및 품질 보증이 필요합니다.
- Human-in-the-loop 검토는 비용이 많이 들거나 사용할 수 없으며 자율 검증이 필요합니다.

이 워크플로는 콘텐츠 생성, 코드 합성 및 검토, 정책 적용, 정렬 검사, 명령 튜닝 및 RAG 사후 처리에 사용됩니다. 또한 지속적인 피드백이 시간이 지남에 따라 더 나은 응답을 형성하여 신뢰할 수 있고 자율적인 의사 결정 루프를 구축하는 데 도움이 되는 자체 개선 에이전트에도 유용합니다.

일반 사용 사례

- 블루 팀 에이전트와 비교한 레드 팀 에이전트
- 코드 또는 계획을 생성, 평가 및 수정하는 에이전트

- 품질 보증, 할루시네이션 감지 및 스타일 적용

기능

- 다양한 모델을 사용하여 분리된 생성 및 평가를 지원합니다(예: Claude for generation 및 Mistral for evaluation).
- 피드백은 구조화되어 있으며 수정된 출력을 표시하는 데 사용됩니다.
- 여러 반복 또는 수렴 임계값 지원

결론

LLMs 최신 소프트웨어 에이전트의 인지적 핵심을 제공하지만 원시 모델 호출만으로는 의도적이고 강력하며 제어 가능한 인텔리전스를 달성하기에 충분하지 않습니다. 출력 생성에서 구조화된 추론 및 목표 정렬 동작으로 이동하려면 모델이 입력을 처리하고 컨텍스트를 관리하고 작업을 조정하는 방법을 정의하는 의도적인 워크플로 패턴에 LLMs를 포함해야 합니다.

LLM 워크플로는 에이전트의 인지 모듈을 구축하기 위한 기반을 도입합니다.

- 프롬프트 체인은 복잡한 추론을 감사 가능한 모듈식 단계로 나눕니다.
- 라우팅을 통해 지능형 작업 분류 및 대상 위임이 가능합니다.
- 병렬화는 처리량을 가속화하고 다양한 추론을 촉진합니다.
- 에이전트 오케스트레이션은 작업 분해 및 역할 기반 실행을 통해 다중 에이전트 협업을 구성합니다.
- 평가자(반사 구체화 루프)를 사용하면 자체 개선, 품질 제어 및 정렬 검사를 수행할 수 있습니다.

각 워크플로는 에이전트의 요구 사항, 작업의 복잡성 및 사용자의 기대치에 맞게 조정할 수 있는 구성 가능한 패턴을 나타냅니다. 이러한 워크플로는 상호 배타적이지 않습니다. 이들은 종종 동적 추론, 다중 에이전트 조정 및 엔터프라이즈급 신뢰성을 지원하는 하이브리드 아키텍처에 결합되는 구성 요소입니다.

에이전트 워크플로 패턴에 대한 다음 장으로 전환하면 이러한 LLM 워크플로가 더 큰 시스템 내에 임베디드 구조로 다시 나타나 목표 위임, 도구 오케스트레이션, 의사 결정 루프 및 수명 주기 자율성을 지원합니다. 이러한 LLM 워크플로를 마스터하는 것은 텍스트뿐만 아니라 추론, 적응 및 의도적으로 행동하는 소프트웨어 에이전트를 설계하는 데 필수적입니다.

에이전트 워크플로 패턴

에이전트 워크플로 패턴은 모듈식 소프트웨어 에이전트를 구조화된 대규모 언어 모델(LLM) 워크플로와 통합하여 자율 추론 및 조치를 가능하게 합니다. 기존 서버리스 및 이벤트 기반 아키텍처에서 영감을 받은 이러한 패턴은 핵심 로직을 정적 코드에서 LLM 증강 에이전트로 전환하여 향상된 적응성과 상황별 의사 결정을 제공합니다. 이 진화는 기존 클라우드 아키텍처를 결정적 시스템에서 동적 해석 및 지능형 증강이 가능한 아키텍처로 변환하는 동시에 확장성 및 응답성의 기본 원칙을 유지합니다.

이 섹션의 내용

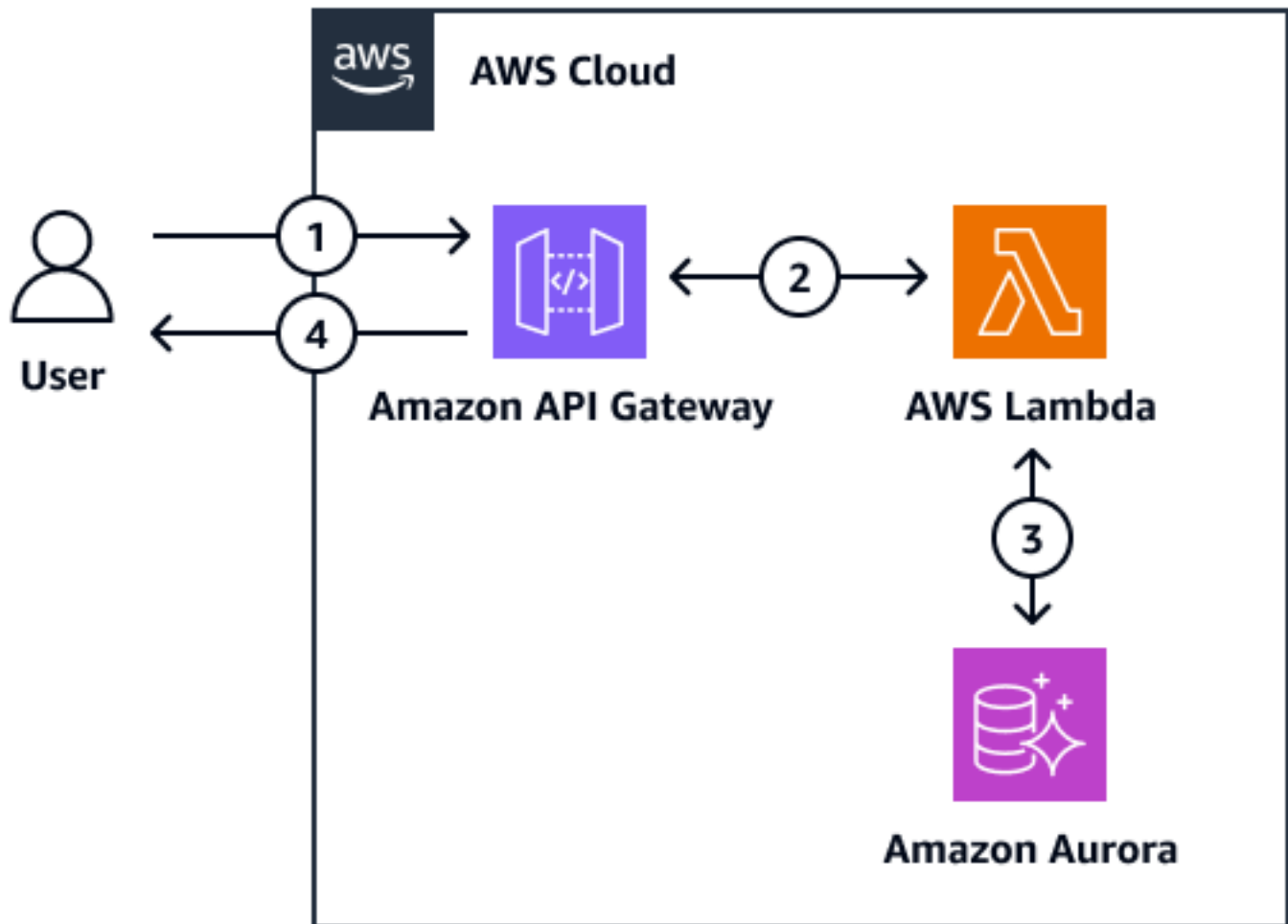
- [이벤트 기반 시스템에서 인지 증강 시스템으로](#)
- [프롬프트 체인 Saga 패턴](#)
- [동적 디스패치 패턴 라우팅](#)
- [병렬화 및 산점 수집 패턴](#)
- [Saga 오케스트레이션 패턴](#)
- [평가자 반사 구체화 루프 패턴](#)
- [에서 에이전트 워크플로 설계 AWS](#)
- [결론](#)

이벤트 기반 시스템에서 인지 증강 시스템으로

최신 클라우드 아키텍처, 특히 서버리스 및 이벤트 기반 원칙을 기반으로 구축된 아키텍처는 전통적으로 라우팅, 팬아웃 및 보강과 같은 패턴에 의존하여 응답성이 뛰어나고 확장 가능한 시스템을 만들었습니다. 에이전트 AI 시스템은 LLM 증강 추론 및 인지 유연성을 중심으로 리프레이밍하면서 이러한 기반을 기반으로 구축됩니다. 이 접근 방식을 사용하면 보다 정교한 문제 해결 및 자동화 기능을 통해 클라우드 환경에서 복잡한 작업을 처리하는 방법을 혁신할 수 있습니다.

이벤트 중심 아키텍처

다음 다이어그램은 일반적인 분산 시스템을 보여줍니다.

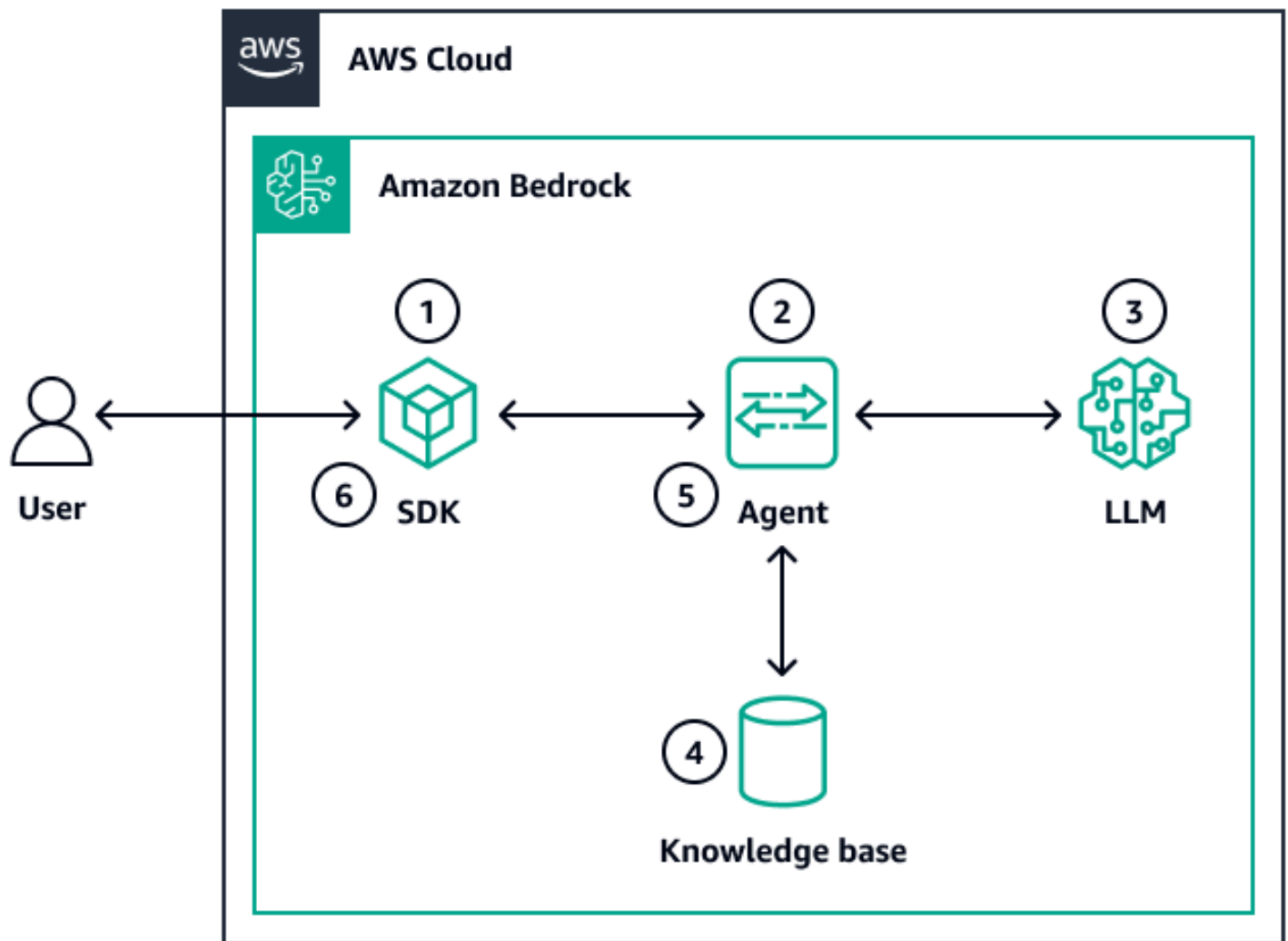


1. 사용자가 Amazon API Gateway에 요청을 제출합니다.
2. Amazon API Gateway는 요청을 AWS Lambda 함수로 라우팅합니다.
3. AWS Lambda 는 Amazon Aurora 데이터베이스를 쿼리하여 데이터 보강을 수행합니다.
4. Amazon API Gateway는 강화된 페이로드를 호출자에게 반환합니다.

이 구조는 안정적이고 확장 가능하지만 기본적으로 정적입니다. 비즈니스 규칙 및 로직 경로는 명시적으로 코딩되어야 하며 변화하는 컨텍스트 또는 불완전한 정보에 적응하는 것은 제한됩니다.

Cognition 증강 워크플로

에이전트 아키텍처는 이벤트 기반 시스템에 인지 증강을 추가합니다. 다음 다이어그램은 에이전트와 동등한 것을 보여줍니다.



1. 사용자가 SDK 또는 API 직접 호출을 통해 쿼리를 제출합니다.
2. Amazon Bedrock 에이전트가 쿼리를 수신합니다.
3. 에이전트는 LLM을 호출하여 쿼리를 해석합니다.
4. 에이전트는 Amazon Bedrock 지식 기반 또는 기타 외부 데이터 소스를 검색하여 의미 체계 보강을 수행합니다.
5. LLM은 컨텍스트가 풍부한 목표 정렬 응답을 합성합니다.
6. 시스템은 합성된 응답을 사용자에게 반환합니다.

이 흐름에서 LLM은 로직을 사용하고, 의도를 이해하고, 관련 컨텍스트를 검색 및 결합한 다음 가장 잘 대응하는 방법을 결정합니다. 이 패턴은 메시지가 추가 라우팅되기 전에 외부 데이터로 보강되는 기존 보강 패턴을 미러링합니다. 그러나 에이전트 시스템에서이 보강은 정적 조희가 아닙니다. 대신 보강은 동적이고 의미론적으로 안내되며 목적에 따라 구동됩니다.

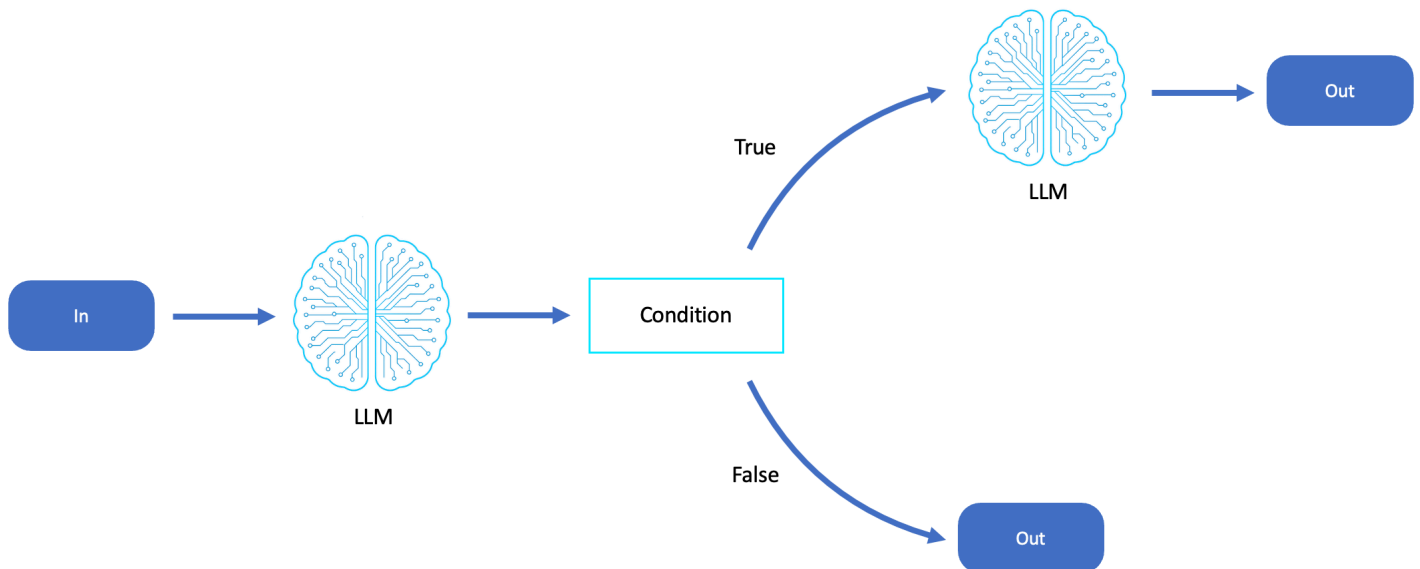
핵심 인사이트

각 LLM 워크플로는 기존의 이벤트 기반 아키텍처 스타일을 미러링하고 발전시키는 에이전트 워크플로 패턴에 매핑할 수 있습니다. 에이전트 워크플로의 기본 구성 요소는 데이터, 도구 및 메모리로 LLM의 컨텍스트를 보강하는 기능입니다. 이렇게 하면 정보에 입각하고 적응력이 뛰어나며 사용자 의도에 맞는 추론 루프가 생성됩니다. 기존 시스템이 조회 데이터로 메시지를 보강하는 경우 에이전트 시스템을 사용하면 소프트웨어가 스크립트처럼 작동하지 않고 지능형 공동 작업자처럼 작동할 수 있습니다.

프롬프트 체인 Saga 패턴

LLM 프롬프트 체인을 이벤트 기반 Saga로 재구상함으로써 워크플로가 자율 에이전트 간에 분산되고 복구 가능하며 의미론적으로 조정되는 새로운 운영 모델을 잠금 해제합니다. 각 프롬프트-응답 단계는 원자성 작업으로 재구성되고, 이벤트로 방출되며, 전용 에이전트가 사용하고, 컨텍스트 메타데이터로 보강됩니다.

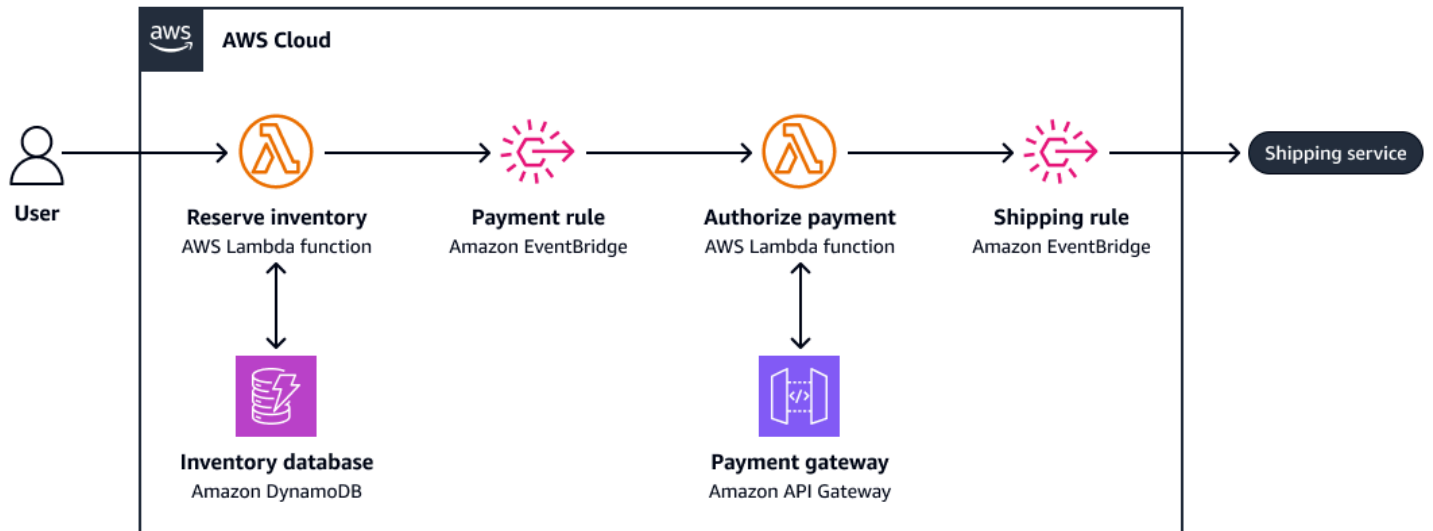
다음 다이어그램은 LLM 프롬프트 체인의 예입니다.



Saga 코레오그래피

Saga 코레오그래피 패턴은 중앙 조정자가 없는 분산 시스템의 구현 접근 방식입니다. 대신 각 서비스 또는 구성 요소는 다음 워크플로 작업을 트리거하는 이벤트를 게시합니다. 이 패턴은 분산 시스템에서 여러 서비스의 트랜잭션을 관리하는 데 널리 사용됩니다. Saga에서 시스템은 일련의 조정된 로컬 트랜잭션을 실행합니다. 실패하면 시스템은 일관성을 유지하기 위해 보정 작업을 트리거합니다.

다음 다이어그램은 Saga 코레오그래피의 예입니다.



1. 인벤토리 예약
2. 결제 권한 부여
3. 배송 주문 생성

3단계에 실패하면 시스템이 보상 작업(예: 결제 취소 또는 인벤토리 릴리스)을 호출합니다.

이 패턴은 서비스가 느슨하게 결합되고 부분적으로 장애가 발생하더라도 시간이 지남에 따라 상태를 일관되게 해결해야 하는 이벤트 기반 아키텍처에서 특히 유용합니다.

프롬프트 체인 패턴

프롬프트 체인은 구조와 목적 모두에서 Saga 패턴과 유사합니다. 컨텍스트를 보존하고 롤백 및 개정을 허용하면서 순차적으로 빌드되는 일련의 추론 단계를 실행합니다.

에이전트 코레오그래피

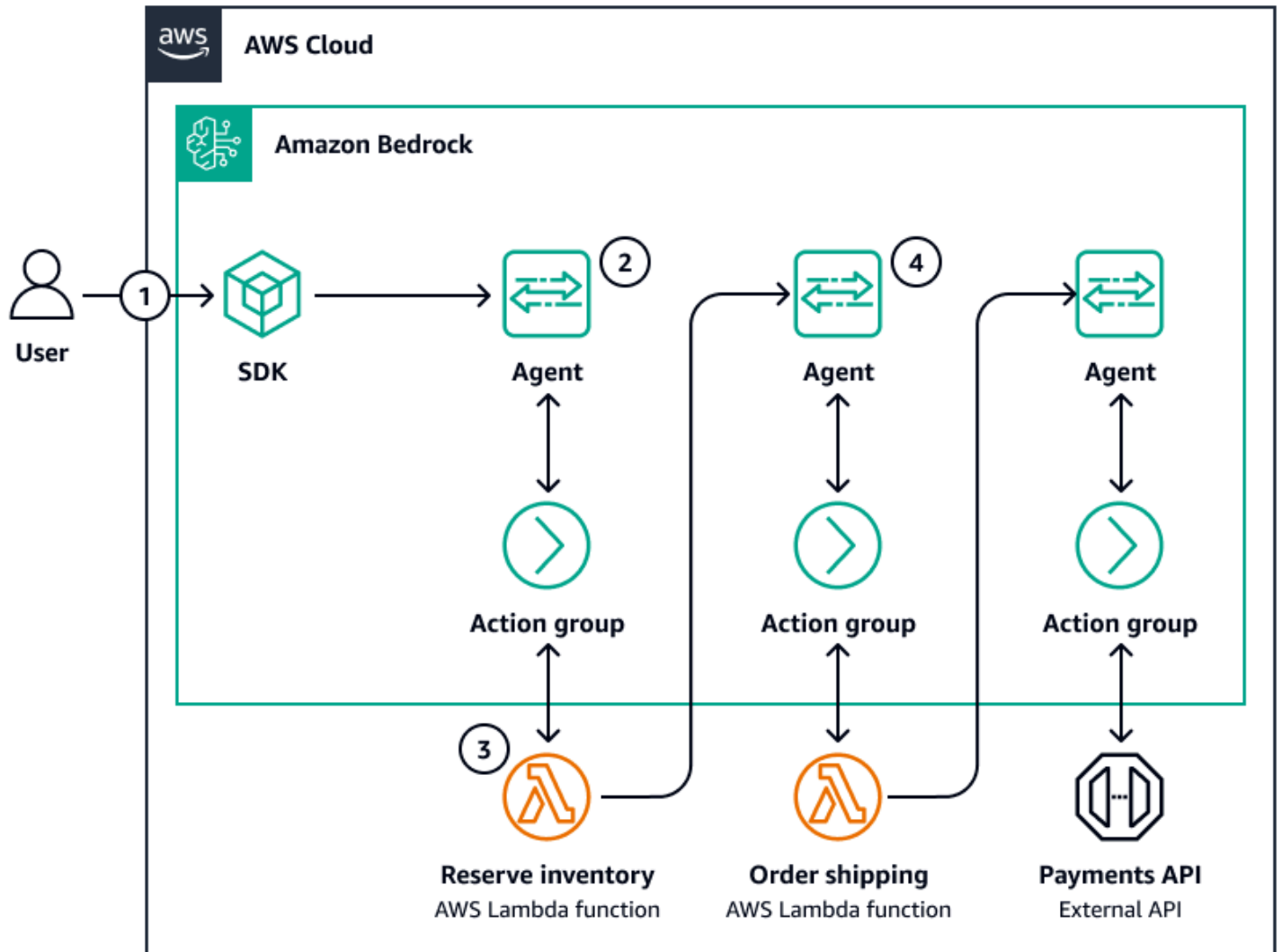
1. LLM은 복잡한 사용자 쿼리를 해석하고 가설을 생성합니다.
2. LLM은 작업을 해결하기 위한 계획을 자세히 설명합니다.
3. LLM은 하위 작업을 실행합니다(예: 도구 호출 사용 또는 지식 검색).
4. LLM은 결과를 만족스럽지 않다고 간주하는 경우 출력을 구체화하거나 이전 단계를 다시 검토합니다.

중간 결과에 결함이 있는 경우 시스템은 다음 중 하나를 수행할 수 있습니다.

- 다른 접근 방식을 사용하여 단계 재시도
- 이전 프롬프트로 되돌리고 다시 계획합니다.
- 평가자 루프(예: 평가자 최적화 패턴)를 사용하여 실패를 감지하고 수정합니다.

Saga 패턴과 마찬가지로 프롬프트 체인을 사용하면 부분 진행 및 롤백 메커니즘을 사용할 수 있습니다. 이는 데이터베이스 트랜잭션을 보정하는 대신 반복적 개선 및 LLM 방향 수정을 통해 발생합니다.

다음 다이어그램은 에이전트 코레오그래피의 예입니다.



1. 사용자가 SDK를 통해 쿼리를 제출합니다.
2. Amazon Bedrock 에이전트는 다음을 통해 추론을 오케스트레이션합니다.
 - 해석(LLM)
 - 계획(LLM)

- 도구 또는 지식 기반을 통한 실행
 - 응답 구성
3. 도구가 실패하거나 불충분한 데이터를 반환하는 경우 에이전트는 작업을 동적으로 다시 계획하거나 다시 설명할 수 있습니다.
 4. 메모리(예: 단기 벡터 스토어)는 여러 단계에서 상태를 유지할 수 있습니다.

요점

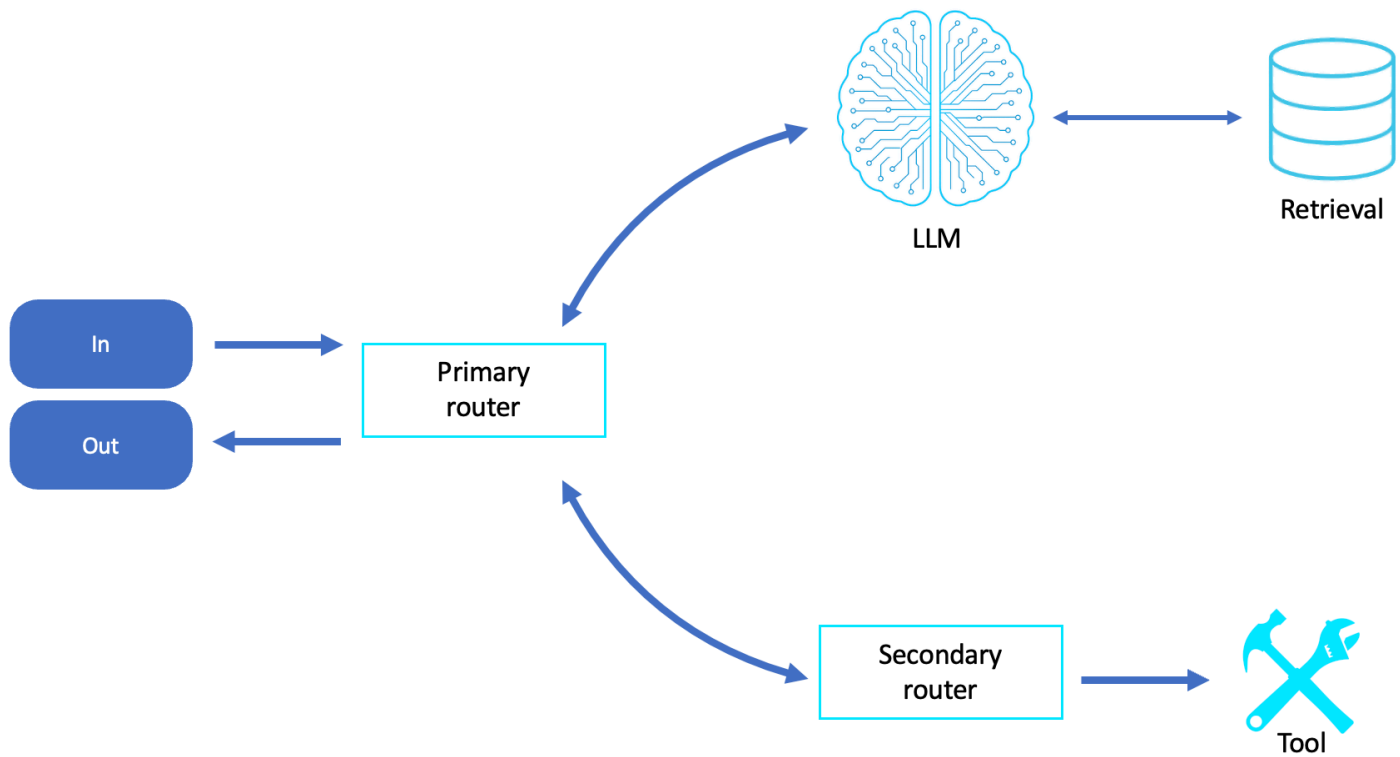
Saga 패턴이 보완 로직을 사용하여 분산 서비스 호출을 관리하는 경우 프롬프트 체인은 반사 시퀀싱 및 적응형 재계획을 통해 추론 작업을 관리합니다. 두 시스템 모두 점진적인 진행 상황, 분산된 결정점 및 장애 복구를 허용하며, 엄격한 롤백이 아닌 정보에 입각한 추론을 통해 이 모든 것을 수행합니다.

프롬프트 체인은 사가스와 인지적으로 동등한 트랜잭션 추론을 도입합니다. 즉, 각 '사고'는 더 광범위한 목표 지향 대화의 일환으로 재평가, 개정 또는 중단됩니다.

동적 디스패치 패턴 라우팅

태스크가 문서 구문 분석부터 자율 소프트웨어 생성에 이르기까지 다양한 최신 에이전트 시스템에서는 요청을 가장 유능한 대규모 언어 모델(LLM) 또는 에이전트로 동적으로 라우팅하는 기능이 중요합니다. 오케스트레이션 스크립트 또는 API 계층에 포함되는 정적 라우팅 로직에는 실시간, 다중 모델, 다중 기능 환경에 필요한 적응성이 부족합니다. 이를 해결하기 위해 LLM 라우팅 워크플로를 동적 디스패치 패턴을 활용하는 이벤트 기반 아키텍처로 변환하여 LLM 호출을 지능적으로 라우팅되는 컨텍스트 인식 이벤트로 변환할 수 있습니다.

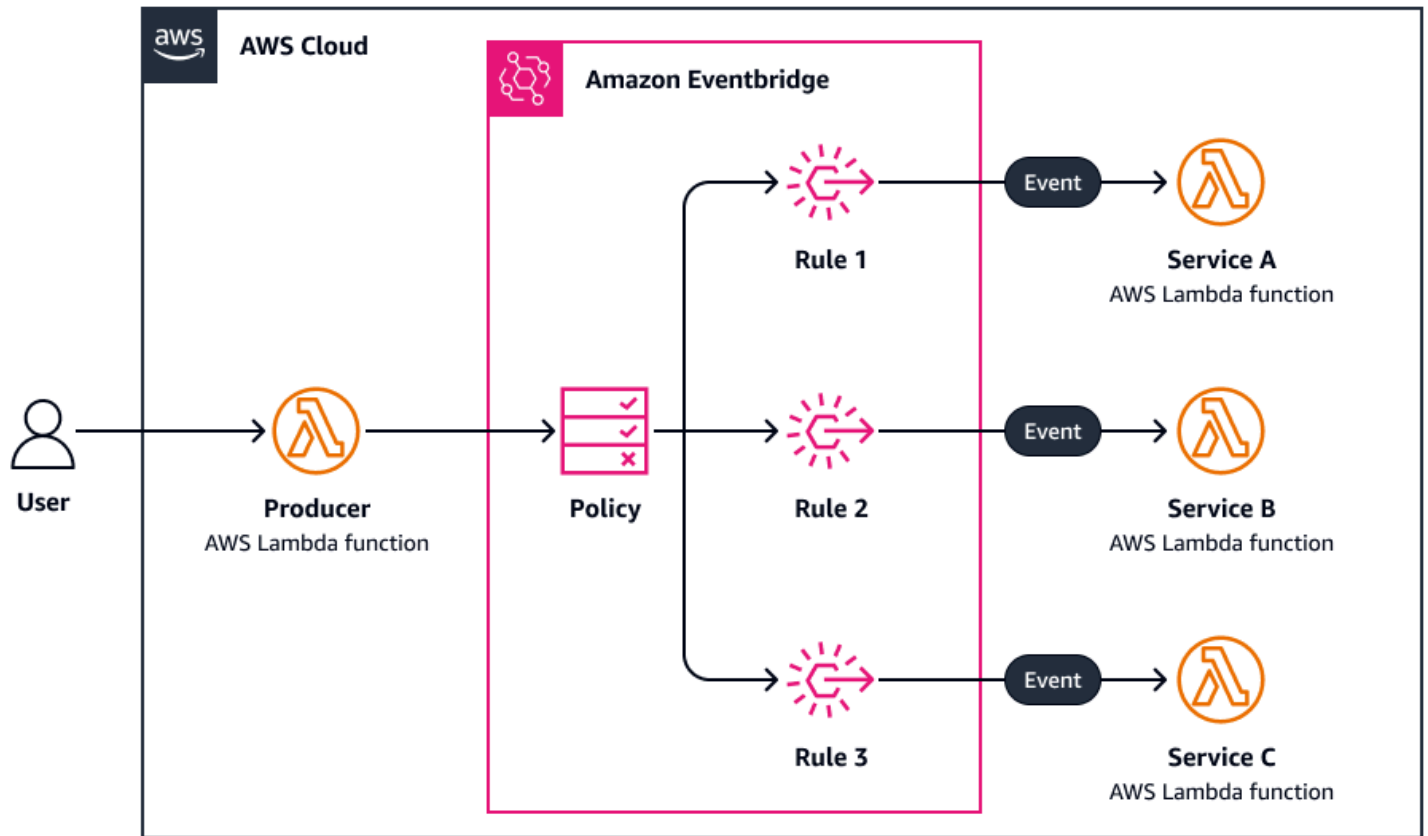
다음 다이어그램은 LLM 라우팅의 예입니다.



동적 디스패치

기존 분산 시스템에서 동적 디스패치 패턴은 이벤트 유형, 소스 및 페이로드와 같은 수신 이벤트 속성을 기반으로 런타임 시 특정 서비스를 선택하고 호출합니다. 이는 일반적으로 수신 이벤트를 평가하고 적절한 대상(예: AWS Lambda 함수 AWS Step Functions 또는 Amazon Elastic Container Service 작업)으로 라우팅할 수 있는 Amazon EventBridge를 사용하여 구현됩니다.

다음 다이어그램은 동적 디스패치의 예입니다.



1. 애플리케이션은 이벤트(예: {"type": "orderCreated", "priority": "high"})를 내보냅니다.
2. Amazon EventBridge는 라우팅 규칙을 기준으로 이벤트를 평가합니다.
3. 이벤트의 속성에 따라 시스템은 다음으로 동적으로 디스패치합니다.
 - HighPriorityOrderProcessor (서비스 A)
 - StandardOrderProcessor (서비스 B)
 - UpdateOrderProcessor (서비스 C)

이 패턴은 느슨한 결합, 도메인 기반 전문화 및 런타임 확장성을 지원합니다. 이를 통해 시스템은 변화하는 요구 사항 및 이벤트 의미에 지능적으로 대응할 수 있습니다.

LLM 기반 라우팅

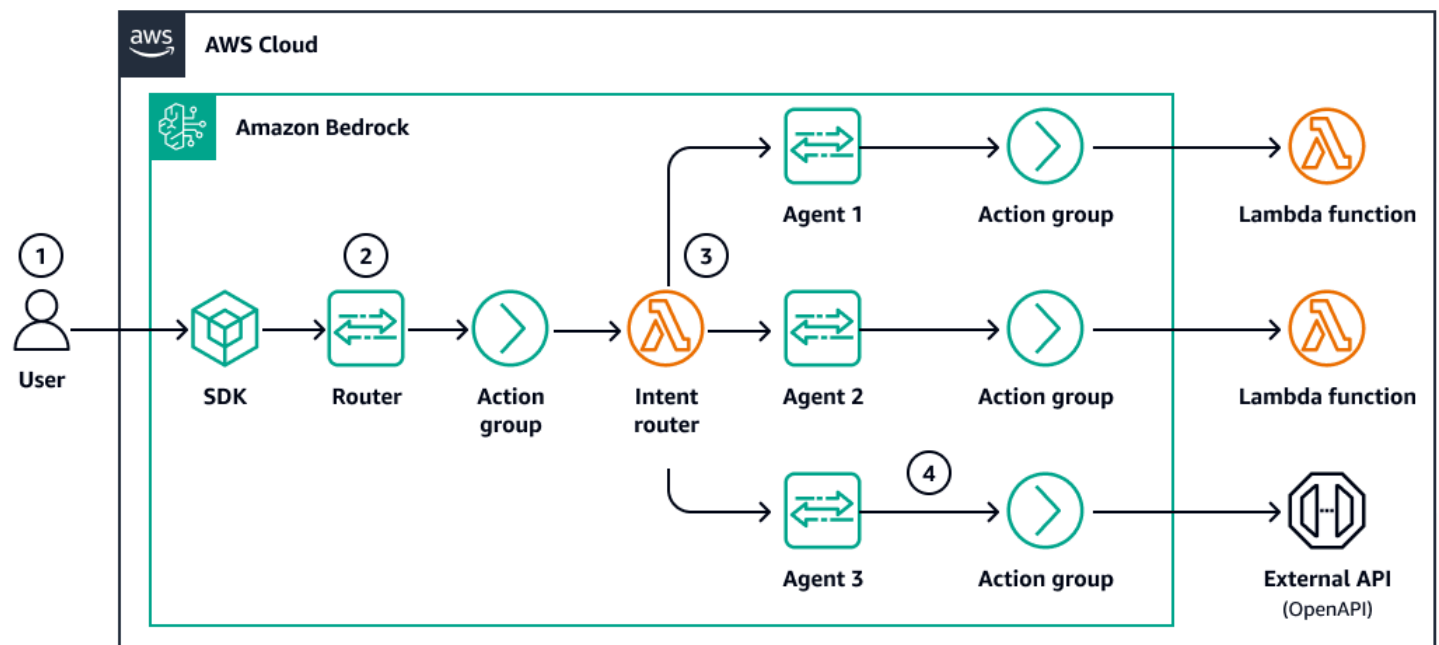
에이전트 시스템에서 라우팅은 동적 작업 위임도 수행하지만, Amazon EventBridge 규칙 또는 메타데이터 필터 대신 LLM은 자연어를 통해 사용자의 의도를 분류하고 해석합니다. 그 결과 유연하고 의미 체계적이며 적응형 형태의 디스패치가 이루어집니다.

에이전트 라우터

이 아키텍처를 사용하면 사전 정의된 스키마 또는 이벤트 유형 없이 풍부한 의도 기반 디스패치가 가능하므로 비정형 입력 및 복잡한 쿼리에 적합합니다.

1. 사용자가 "내 계약 조건을 검토할 수 있나요?" 요청을 제출합니다.
2. LLM은 이를 법적 문서 작업으로 해석합니다.
3. 에이전트는 작업을 다음 중 하나 이상으로 라우팅합니다.
 - 계약 검토 프롬프트 템플릿
 - 법적 추론 하위 에이전트
 - 문서 구문 분석 도구

다음 다이어그램은 에이전트 라우터의 예입니다.



1. 사용자가 SDK를 통해 자연어 요청을 제출합니다.
2. Amazon Bedrock 에이전트는 LLM을 사용하여 작업을 분류합니다(예: 법률, 기술 또는 일정).
3. 에이전트는 작업 그룹을 통해 작업을 동적으로 라우팅하여 필요한 에이전트를 호출합니다.
 - 도메인별 에이전트
 - 전문화된 도구 체인
 - 사용자 지정 프롬프트 구성

4. 선택한 핸들러가 작업을 처리하고 맞춤형 응답을 반환합니다.

요점

기존 동적 디스패치가 구조화된 이벤트 속성을 기반으로 라우팅하기 위해 Amazon EventBridge 규칙을 사용하는 경우 에이전트 라우팅은 LLMs 사용하여 의미와 의도를 기반으로 작업을 의미상 분류하고 라우팅합니다. 이렇게 하면 다음을 활성화하여 시스템의 유연성이 향상됩니다.

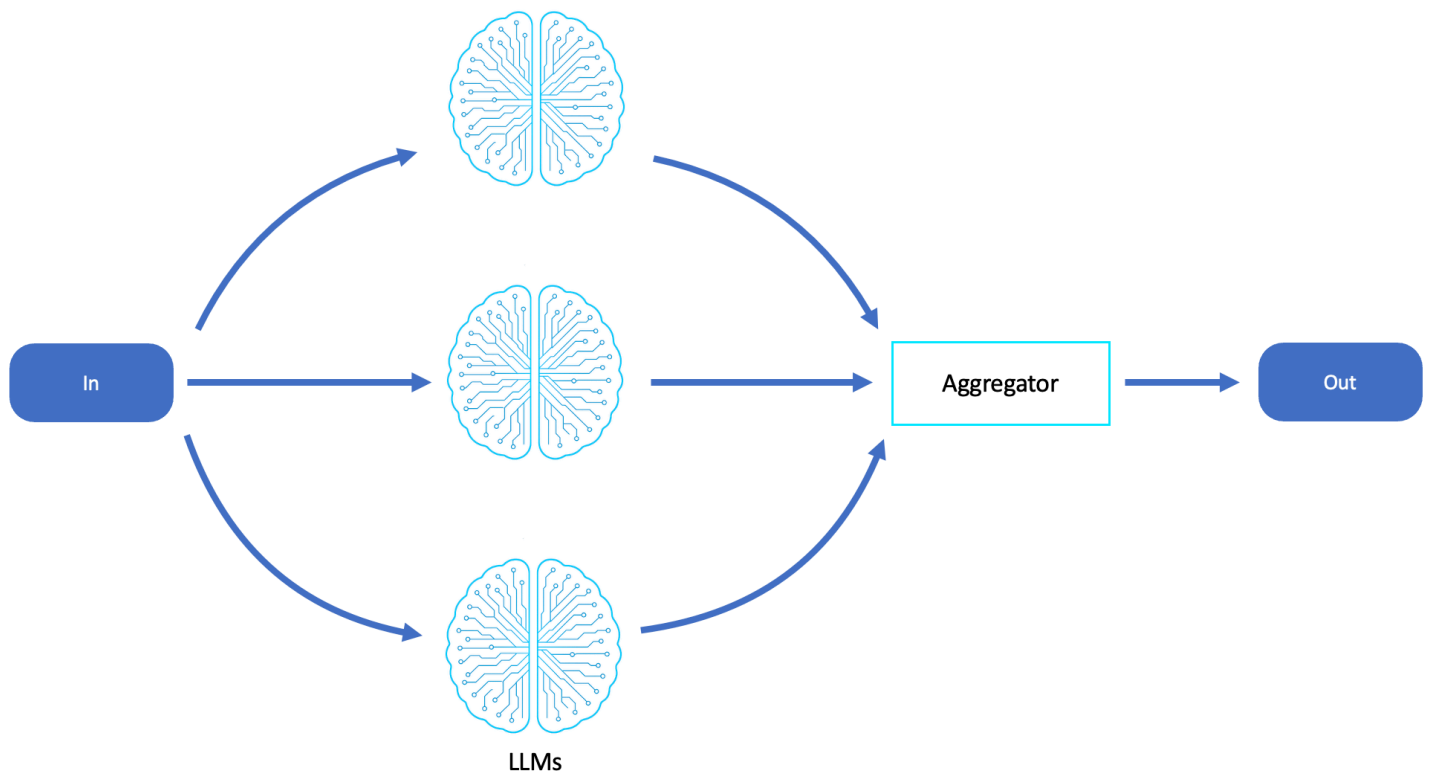
- 더 광범위한 입력 이해
- 지능형 대체 및 도구 선택
- 새로운 에이전트 역할 또는 프롬프트 스타일을 통한 자연스러운 확장성

에이전트 라우팅은 엄격한 규칙을 동적 인지 디스패치로 대체하므로 시스템이 코드가 아닌 언어에 따라 진화할 수 있습니다.

병렬화 및 산점 수집 패턴

대규모 문서 요약, 여러 솔루션 경로 평가, 다양한 관점 비교와 같은 많은 고급 추론 및 생성 작업은 프롬프트의 병렬 실행의 이점을 누릴 수 있습니다. 확장성, 응답성 및 내결함성이 필요한 경우 기존 순차 워크플로는 부족합니다. 이를 해결하기 위해 이벤트 기반 산점 수집 패턴을 사용하여 LLM 기반 병렬화를 재구성할 수 있습니다. 이 패턴에서는 작업이 자율 에이전트로 동적으로 팬아웃되고 결과가 지능적으로 합성됩니다.

다음 다이어그램은 LLM 병렬화 워크플로의 예입니다.



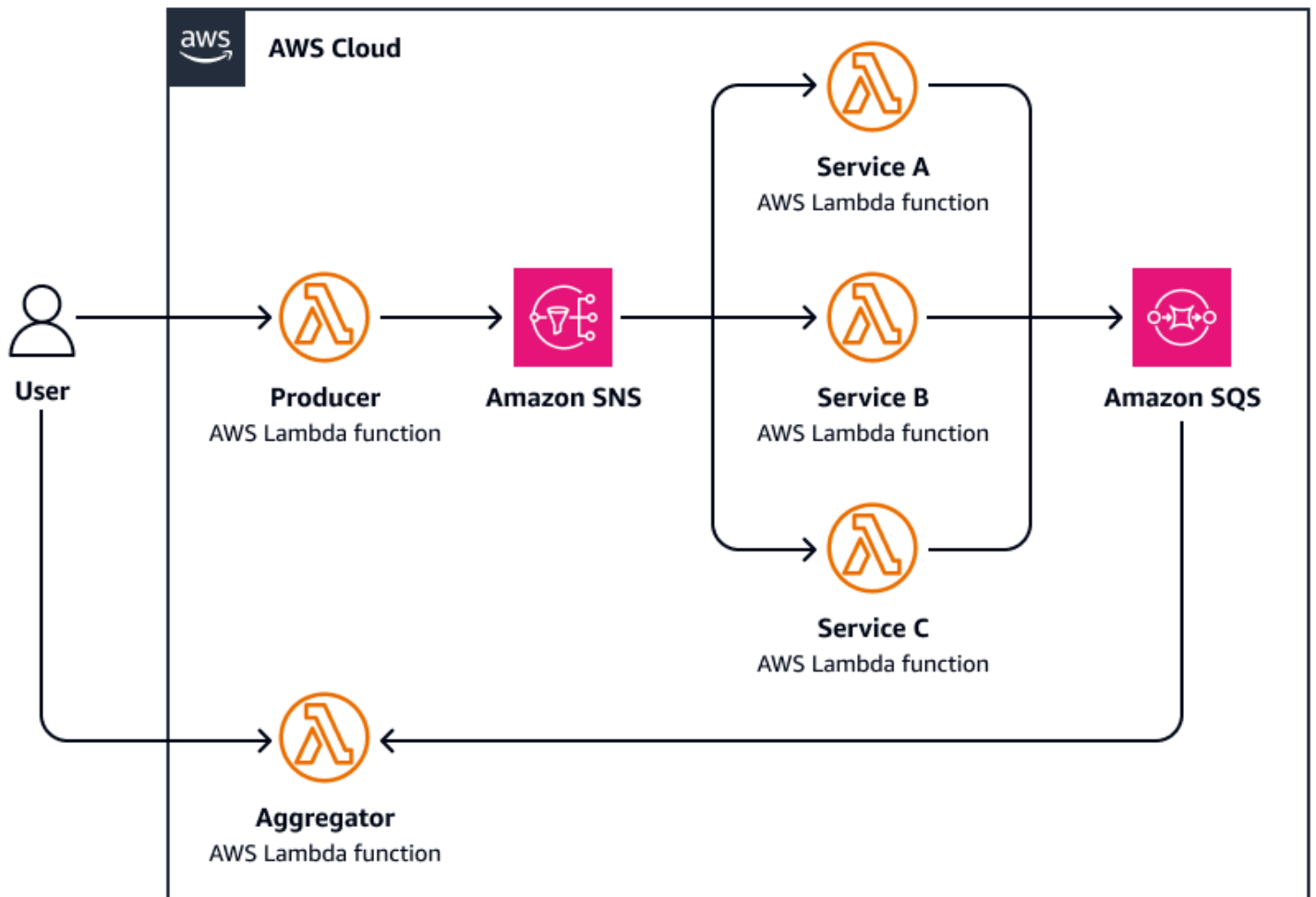
확산 및 수집

분산 시스템에서 산점도 수집 패턴은 여러 서비스 또는 처리 장치에 작업을 병렬로 전송하고 응답을 기다린 다음 결과를 통합 출력으로 집계합니다. 팬아웃과 달리 산점도 수집은 응답을 예상하고 일반적으로 결과를 결합, 비교 및 선택하는 로직을 적용하기 때문에 조정됩니다.

병렬화 및 산점도 수집을 위한 일반적인 구현은 다음과 같습니다.

- AWS Step Functions 병렬 작업 실행을 위한 상태 매핑
- AWS Lambda 동시성을 사용하여 여러 호출된 함수의 결과 조정
- 상관관계 IDs 및 집계 워크플로가 있는 Amazon EventBridge
- Amazon Simple Storage Service(Amazon S3), Amazon DynamoDB 또는 대기열을 사용하여 팬아웃을 관리하고 결과를 수집하는 사용자 지정 컨트롤러 패턴

다음 다이어그램은 산점도 수집의 예입니다.



1. 사용자는 Amazon Simple Notification Service(Amazon SNS) 주제에 병렬 메시지를 게시하여 작업을 분산시키는 중앙 조정자 함수에 요청을 보냅니다.
2. 각 메시지에는 작업 메타데이터가 포함되어 있으며 전문 작업자에게 라우팅됩니다 AWS Lambda.
3. 각 작업자는 할당된 하위 작업(예: 외부 API 쿼리, 문서 처리, 데이터 분석)을 AWS Lambda 독립적으로 처리합니다.
4. 결과는 Amazon Simple Queue Service(Amazon SQS)와 같은 공통 스토리지 계층에 기록됩니다.
5. 애그리게이터 함수는 모든 응답이 완료될 때까지 기다린 다음 다음 다음을 수행합니다.
 - 결과를 수집하고 집계합니다(예: 요약 병합, 가장 적합한 일치 항목 선택).
 - 최종 응답을 보내거나 다운스트림 워크플로를 트리거합니다.

산점도 수집 패턴의 일반적인 사용 사례는 다음과 같습니다.

- 페더레이션 검색

- 가격 비교 엔진
- 집계된 데이터 분석
- 다중 모델 추론

LLM 기반 병렬화(산점 수집 인식)

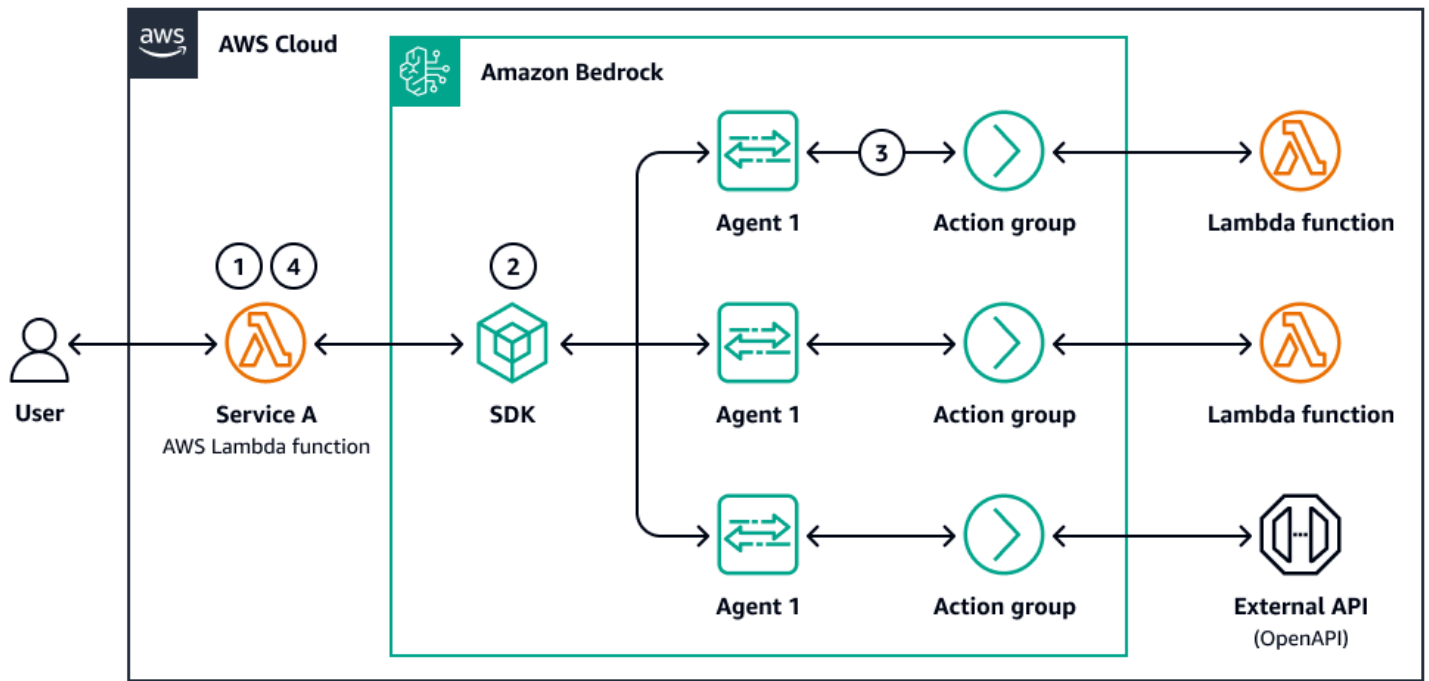
에이전트 시스템에서 병렬화는 하위 작업을 여러 LLM 호출 또는 에이전트에 분산하여 산란 수집을 면밀히 모니터링하며, 각각 독립적으로 문제의 일부를 추론합니다. 반환된 결과는 종종 다른 LLM 또는 컨트롤러 에이전트인 집계 프로세스에 의해 수집되고 합성됩니다.

에이전트 병렬화

1. 에이전트가 "이 10개 보고서에 대한 인사이트 요약" 요청을 제출합니다.
2. 보고서를 10개의 병렬 LLM 요약 작업으로 분산합니다.
3. 모든 요약을 반환하면 에이전트는 다음을 수행합니다.
 - 요약을 통합 브리핑으로 집계합니다.
 - 테마 또는 모순 식별
 - 합성된 출력을 사용자에게 전송합니다.

이 에이전트 워크플로는 확장 가능한 모듈식 적응형 병렬 추론을 지원합니다. 이는 높은 인지 처리량이 필요한 사용 사례에 적합합니다.

다음 다이어그램은 에이전트 병렬화의 예입니다.



1. 사용자가 멀티파트 쿼리 또는 문서 세트를 제출합니다.
2. 컨트롤러 AWS Lambda 또는 단계 함수는 하위 작업을 배포합니다. 각 작업은 자체 프롬프트를 사용하여 Amazon Bedrock LLM 호출 또는 하위 에이전트를 호출합니다.
3. 호출 및 하위 작업이 완료되면 결과가 저장되고(예: Amazon S3 또는 메모리 스토어) 집계 단계에서 출력을 병합, 비교 또는 필터링합니다.
4. 시스템은 사용자 또는 다운스트림 에이전트에게 최종 응답을 반환합니다.

이 시스템에는 추적성, 내결함성 및 선택적 결과 가중치 또는 선택 로직이 포함된 분산 추론 루프가 있습니다.

요점

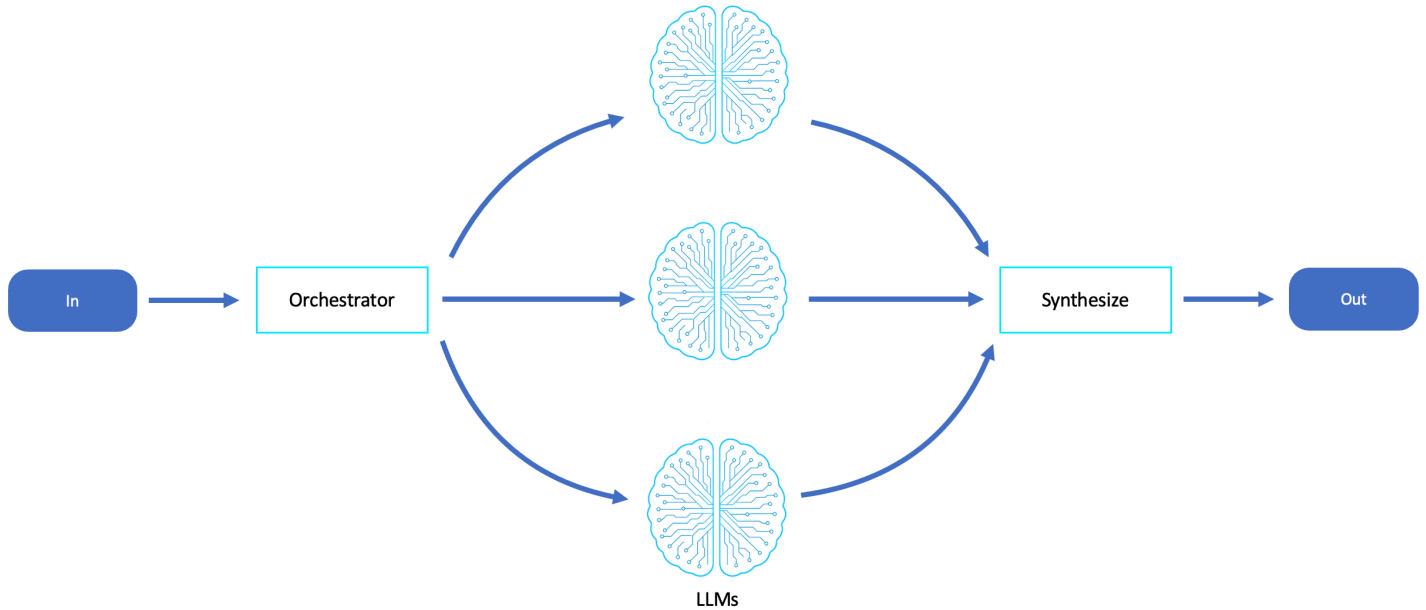
에이전트 병렬화는 산점도 수집 패턴을 사용하여 LLM 작업을 분산하므로 병렬 처리 및 지능형 결과 합성이 가능합니다.

Saga 오케스트레이션 패턴

프롬프트 체인, 데이터 처리 단계, 도구 호출 및 에이전트 협업에 걸쳐 LLMs으로 구동되는 워크플로가 점점 더 복잡해짐에 따라 지능형 오케스트레이션의 필요성이 필수적입니다. 긴밀하게 결합된 스크립트 또는 미리 결정된 정적 실행 흐름에 의존하는 대신 이러한 워크플로를 이벤트 기반 오케스트레이션

패턴으로 구현하여 LLM 기반 시스템이 자율 에이전트 간에 다단계 작업을 동적으로 조정, 모니터링 및 조정할 수 있습니다.

다음 다이어그램은 오케스트레이터의 예입니다.



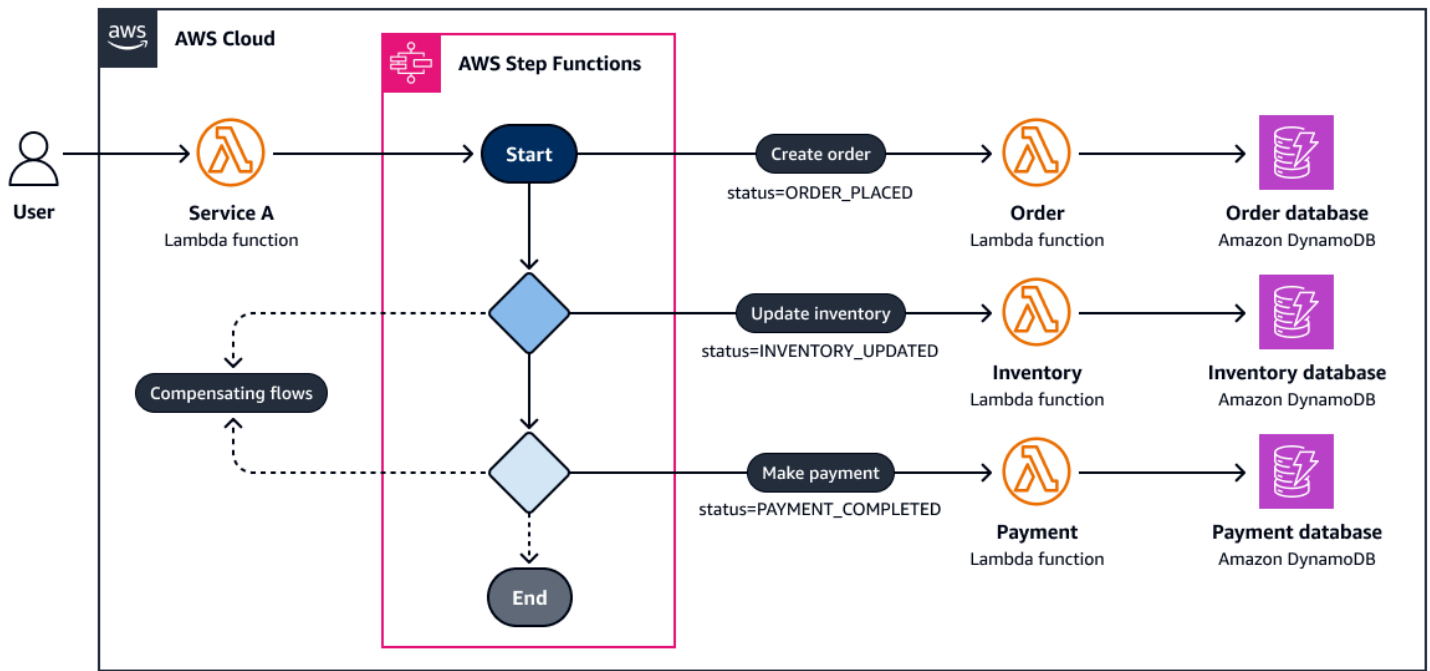
이벤트 오케스트레이션

기존 분산 시스템에서 이벤트 오케스트레이션은 중앙 코디네이터가 여러 서비스 또는 작업에 대한 제어 흐름을 명시적으로 지시하여 복잡한 워크플로를 관리하는 패턴을 말합니다. 이벤트 코레오그래피 (각 서비스가 독립적으로 반응하는 경우)와 달리 오케스트레이션은 전체 프로세스에 대한 중앙 집중식 로직, 가시성 및 제어를 제공합니다.

이는 일반적으로 다음 도구를 사용하여 구현됩니다.

- AWS Step Functions - 상태 저장 워크플로 정의 및 실행
- AWS Lambda - 오케스트레이션된 흐름 내에서 개별 작업 수행
- Amazon SQS 또는 Amazon EventBridge - 비동기 단계 또는 응답을 트리거합니다.

다음 다이어그램은 Saga 오케스트레이션의 예입니다.



AWS Step Functions 워크플로는 고객 주문 프로세스를 관리합니다.

1. 주문 생성(AWS Lambda)
2. 인벤토리 업데이트(AWS Lambda)
3. 결제(AWS Lambda)

오케스트레이터는 재시도, 병렬 브랜치, 제한 시간 및 실패를 관리하여 각 단계를 조정합니다.

역할 기반 에이전트 시스템(오케스트레이터)

에이전트 시스템에서 오케스트레이터 패턴은 이벤트 오케스트레이션을 미러링하지만 각각 역할 또는 전문화가 정의된 여러 추론 에이전트에 로직을 분산합니다. 중앙 오케스트레이터 에이전트는 전체 작업을 해석하고, 하위 작업으로 분해하고, 작업자 에이전트에게 위임합니다. 각 에이전트는 특정 도메인(예: 연구, 코딩, 요약, 검토)에 최적화되어 있습니다.

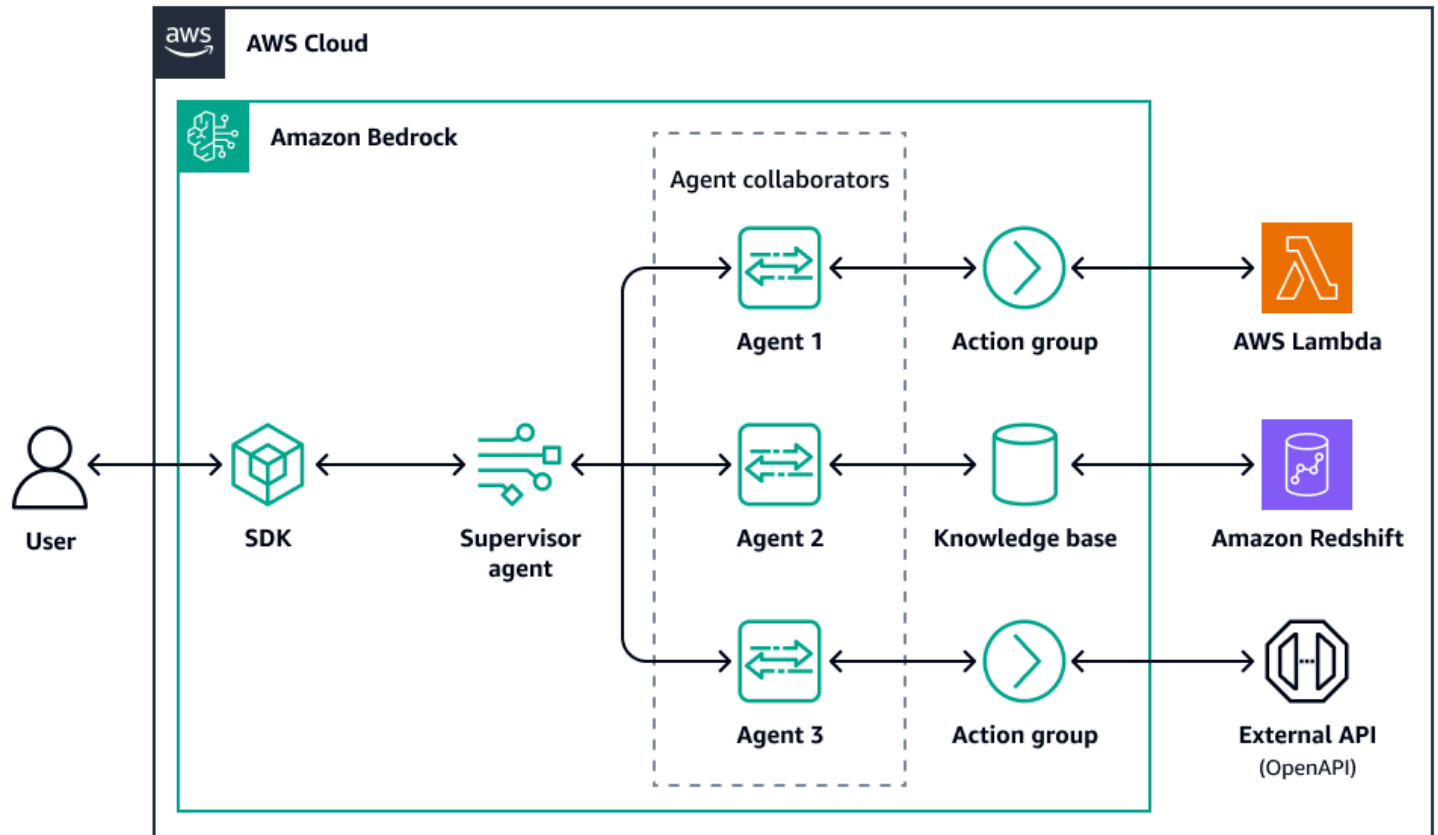
Supervisor

1. 사용자가 “프로젝트 개요 생성 및 상위 5개 경쟁자 요약” 쿼리를 제출합니다.
2. 오케스트레이터 에이전트는 다음을 수행합니다.
 - 연구 에이전트를 할당하여 경쟁자 데이터를 찾습니다.
 - 원시 조사 결과를 요약 에이전트로 보냅니다.

- 결과를 간이 라이터 에이전트에 전달합니다.
- 사용자의 최종 출력을 컴파일합니다.

각 에이전트는 독립적으로 작동하지만 오케스트레이터는 작업을 조정합니다. 이는 워크플로 작업을 처리하는 Lambda 함수와 같습니다.

다음 다이어그램은 감독자의 예입니다.



1. 사용자가 Amazon Bedrock 감독자 에이전트에게 작업을 제출합니다.
2. 감독자 에이전트는 요청을 각 에이전트 공동 작업자의 하위 작업으로 구문 분석합니다.
3. 각 하위 작업은 역할별 프롬프트 또는 도구 체인을 사용하여 공동 작업자 에이전트에 할당됩니다.
4. 작업자 에이전트는 작업 그룹을 통해 외부 APIs 또는 도구를 호출합니다.
5. 각 작업자 에이전트는 출력을 구조화된 형식으로 반환합니다.
6. 모든 작업자가 결과를 반환하면 감독자는 최종 응답을 평가, 합성 및 반환합니다.

이 구조를 사용하면 복잡한 다단계 에이전트 워크플로에서 모듈성, 적응성 및 내부 검사를 수행할 수 있습니다.

요점

이벤트 오케스트레이션이 중앙 집중식 제어(예: AWS Step Functions)를 사용하여 서비스 실행을 지시하는 경우 역할 기반 에이전트 시스템은 LLM 기반 오케스트레이터 에이전트를 사용하여 목표를 추론하고, 하위 작업을 작업자 에이전트에게 위임하고, 최종 출력을 합성합니다.

두 패러다임 모두에서 오케스트레이터는 다음을 수행합니다.

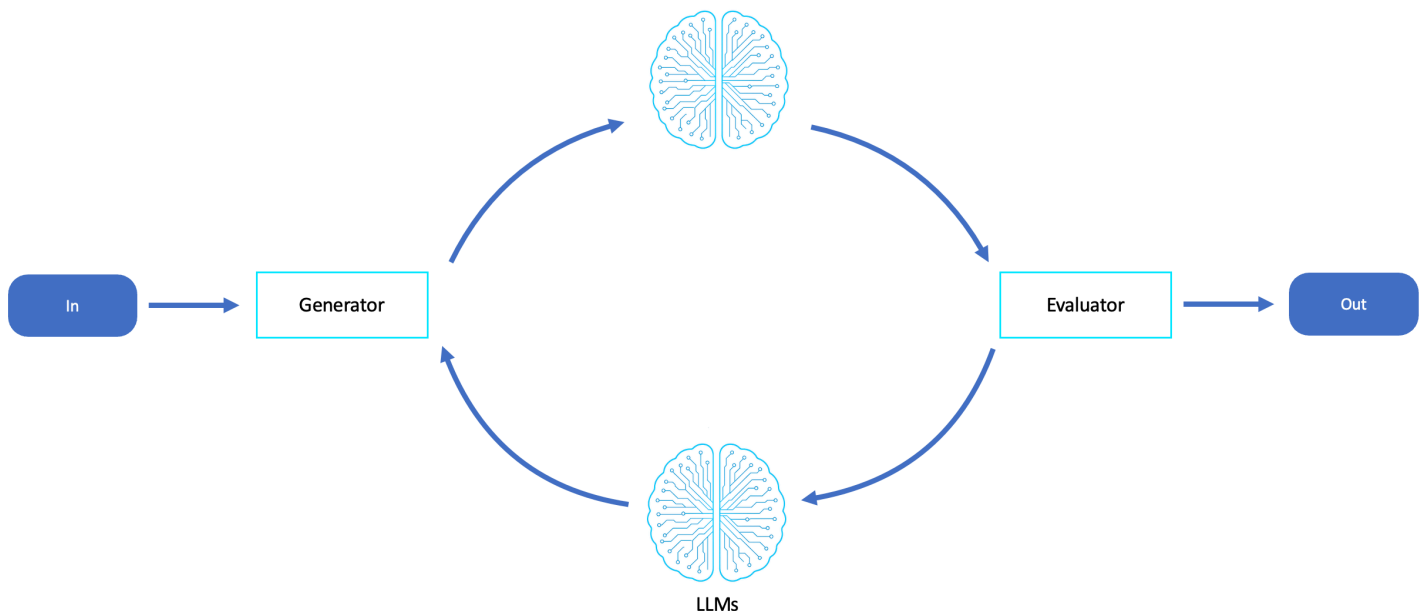
- 컨텍스트 및 실행 흐름을 유지합니다.
- 분기, 시퀀싱 및 오류 처리를 처리합니다.
- 분산 구성 요소에서 통합 결과를 생성합니다.

그러나 에이전트 오케스트레이션에는 추론, 적응성 및 의미 체계 위임이 추가됩니다. 따라서 개방형의 모호하고 진화하는 작업에 적합합니다.

평가자 반사 구체화 루프 패턴

코드 생성, 요약 또는 자율 의사 결정과 같은 작업은 런타임 피드백의 이점을 크게 활용하므로 시스템이 관찰 및 개선을 통해 발전할 수 있습니다. 이를 운영하기 위해 반사 세분화 주기를 자율적이고 지능적인 워크플로에 맞게 조정된 시스템 엔지니어링에서 영감을 받은 패턴인 이벤트 기반 피드백 제어 루프로 구현할 수 있습니다.

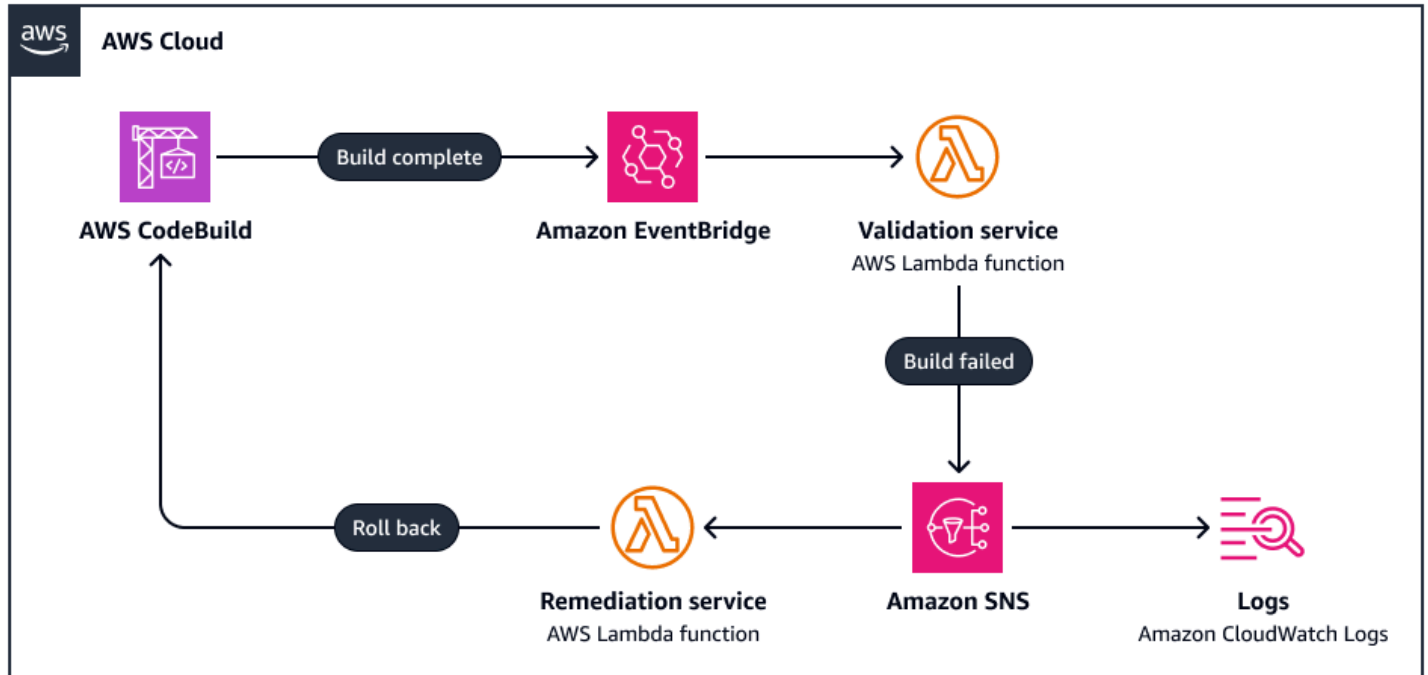
다음 다이어그램은 평가자 반사 구체화 피드백 루프의 예입니다.



피드백 제어 루프

피드백 제어 루프는 자체 출력 및 동작을 모니터링하고 정의된 기준 또는 원하는 상태를 기준으로 이를 평가한 다음 그에 따라 작업을 조정하는 패턴입니다. 이 아키텍처는 제어 이론에서 영감을 받았으며 자동화, 지속적 통합 및 지속적 전달(CI/CD) 파이프라인, 기계 학습 운영과 같은 도메인의 기본입니다.

다음 다이어그램은 피드백 제어 루프의 예입니다.



1. 배포 파이프라인은 buildComplete 이벤트를 내보냅니다.
2. 이벤트는 빌드를 검증하는 자동 테스트 또는 평가 작업을 트리거합니다.
3. 검증에 실패하면(예: 테스트 실패, 보안 문제 또는 정책 위반으로 인해) 시스템은 다음을 수행합니다.
 - buildComplete 이벤트를 내보냅니다.
 - 문제를 로깅하거나 알림을 보냅니다.
 - 롤백, 패치 적용 또는 재시도와 같은 수정 또는 수정 작업을 트리거합니다.

루프는 허용 가능한 결과 또는 에스컬레이션이 생성되거나 제한 시간이 발생할 때까지 계속됩니다. 이 패턴은 일반적으로 다음과 같은 경우에 사용됩니다.

- 이벤트를 평가 또는 수정 작업으로 라우팅하는 Amazon EventBridge 규칙
- AWS Step Functions 반복 재시도 로직 및 평가 결과에 대한 분기

- 피드백 트리거 및 알림에 대한 Amazon Simple Notification Service(Amazon SNS) 또는 Amazon CloudWatch 경보
- AWS Lambda 수정 조치를 적용하기 위한 함수 또는 컨테이너화된 작업자

피드백 제어 루프(평가자)

평가자 워크플로는 LLMs 또는 추론 에이전트로 구동되는 인지 피드백 루프입니다. 프로세스는 다음으로 구성됩니다.

1. 생성기 에이전트 또는 LLM은 출력(예: 계획, 답변 또는 초안)을 생성합니다.
2. 평가자 에이전트는 비평 프롬프트 또는 평가 마찰을 사용하여 결과를 검토합니다.
3. 피드백을 기반으로 원래 에이전트 또는 새 옵티마이저 에이전트가 출력을 수정합니다.

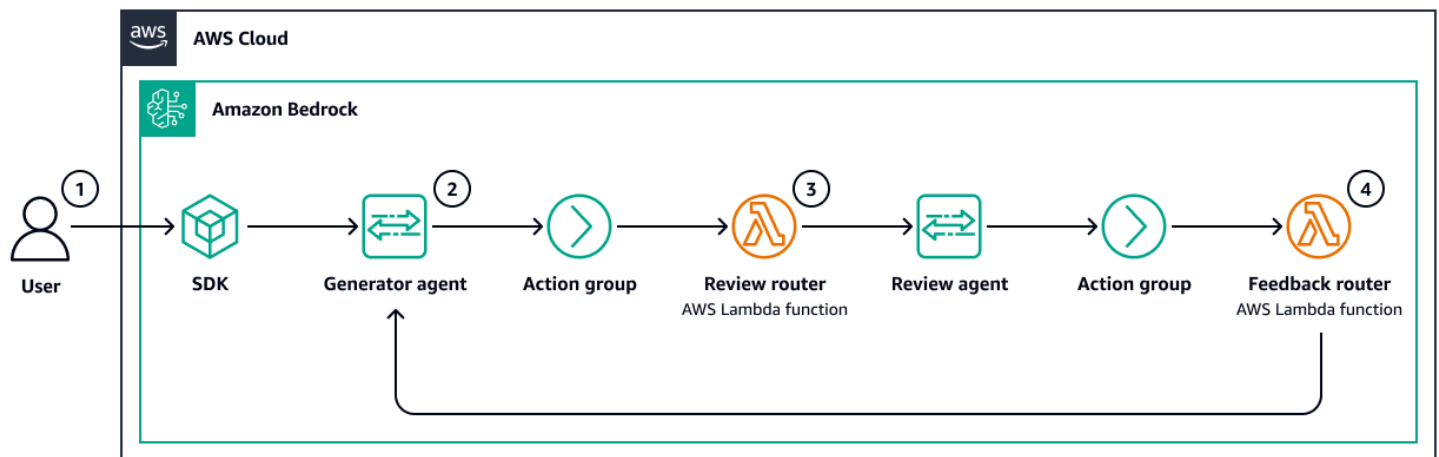
루프는 결과가 기준 집합을 충족하거나 승인되거나 재시도 제한에 도달할 때까지 반복됩니다.

평가자

1. 사용자가 에이전트에게 정책 요약을 작성하도록 요청합니다.
2. 생성기 에이전트가 초안을 작성합니다.
3. 평가자 에이전트는 적용 범위, 어조 및 법적 정확성을 확인합니다.
4. 응답이 부적절한 경우 피드백 루프가 수렴될 때까지 수정되고 다시 제출됩니다.

이를 통해 인적 입력 없이 자체 평가, 반복적 구체화 및 적응형 출력 제어가 가능합니다.

다음 다이어그램은 피드백 제어 루프(평가자)의 예입니다.



1. 사용자가 작업을 실행합니다(예: 비즈니스 전략 초안 작성).
2. Amazon Bedrock 에이전트는 LLM을 사용하여 초기 초안을 생성합니다.
3. 두 번째 에이전트(또는 후속 프롬프트)는 구조화된 평가(예: "명확성, 완전성 및 어조로이 출력 평가")를 수행합니다.
4. 등급이 임계값 아래로 떨어지면 다음과 같이 응답이 수정됩니다.
 - 임베디드 비평으로 생성기 다시 호출
 - 특수 정련소 에이전트에게 피드백 전송
 - 허용 가능한 응답에 도달할 때까지 반복

AWS Lambda 컨트롤러 또는와 같은 선택적 구성 요소는 피드백 임계값, 재시도 및 대체 전략을 관리할 AWS Step Functions 수 있습니다.

요점

기존 피드백 제어 루프가 이벤트, 지표 및 수정 로직을 사용하여 시스템 동작을 검증하고 조정하는 경우 에이전트 평가자 루프는 추론 에이전트를 사용하여 출력을 동적으로 평가, 반영 및 수정합니다.

두 패러다임 모두에서:

- 출력은 생성된 후 평가됩니다.
- 피드백을 기반으로 수정 또는 개선 작업이 트리거됩니다.
- 시스템이 대상 품질 또는 목표에 지속적으로 적응함

에이전트 버전은 정적 검증을 의미론적 반영으로 변환하여 자체 효과를 평가하는 자체 개선 에이전트를 지원합니다.

에서 에이전트 워크플로 설계 AWS

이 가이드의 각 패턴은를 사용하여 빌드할 수 있습니다 AWS 서비스. Amazon Bedrock 에이전트는 오케스트레이션, 데이터 액세스 및 상호 작용 채널을 제공합니다.

구성 요소	AWS 서비스	용도
LLM 추론	Amazon Bedrock	에이전트 로직, 계획, 도구 사용

도구 실행	AWS Lambda, Amazon ECS, Amazon SageMaker	에이전트를 위한 외부 도구 호스팅
메모리 및 RAG	Amazon Bedrock 지식 기반, Amazon S3, OpenSearch	영구 및 의미 체계 메모리
오케스트레이션	AWS Step Functions	다단계 작업 및 에이전트 조정
이벤트 라우팅	Amazon EventBridge, Amazon SQS	분리된 에이전트 간 메시징
사용자 인터페이스	Amazon API Gateway, AWS AppSync, SDK	애플리케이션 또는 시스템의 진입점
모니터링	Amazon CloudWatch, AWS X-Ray, AWS Distro for OpenTelemetry	관찰성 및 에이전트 내부 검사

결론

에이전트 워크플로 패턴은 이벤트 기반 아키텍처의 차세대 진화 단계로, 비즈니스 로직은 정적으로 정의되지 않지만 대규모 언어 모델(LLM) 강화 인식을 사용하여 동적으로 추론됩니다. 기존의 클라우드 네이티브 프리미티브와 LLM 워크플로 및 에이전트 설계 패턴을 결합하여 조직은 목적에 따라 대응하고 경험을 통해 학습하는 적응형 지능형 모듈식 시스템을 구축할 수 있습니다.

이러한 패턴에서 Amazon Bedrock은 에이전트 인식의 게이트웨이로, LLM 기반 에이전트가 이벤트 워크플로에 액세스하고 도구 및 메모리와 상호 작용하며 구조화되고 추적 가능하며 정렬된 결과를 제공할 수 있도록 합니다.

에이전트 시스템을 설계하고 배포할 때 이러한 워크플로 패턴은 구성 가능한 자율 AI 아키텍처를 구축하기 위한 청사진을 제공합니다. 이러한 시스템은 서버리스 모범 사례를 기반으로 하며 지능형 파운데이션 모델로 보강됩니다.

문서 기록

아래 표에 이 가이드의 주요 변경 사항이 설명되어 있습니다. 향후 업데이트에 대한 알림을 받으려면 [RSS 피드](#)를 구독하십시오.

변경 사항	설명	날짜
최초 게시	—	2025년 7월 14일

기계 번역으로 제공되는 번역입니다. 제공된 번역과 원본 영어의 내용이 상충하는 경우에는 영어 버전이 우선합니다.