



Modelli e flussi di lavoro di intelligenza artificiale agentica su AWS

AWS Linee guida prescrittive



AWS Linee guida prescrittive: Modelli e flussi di lavoro di intelligenza artificiale agentica su AWS

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

I marchi e l'immagine commerciale di Amazon non possono essere utilizzati in relazione a prodotti o servizi che non siano di Amazon, in una qualsiasi modalità che possa causare confusione tra i clienti o in una qualsiasi modalità che denigri o discrediti Amazon. Tutti gli altri marchi non di proprietà di Amazon sono di proprietà dei rispettivi proprietari, che possono o meno essere affiliati, collegati o sponsorizzati da Amazon.

Table of Contents

Introduzione	1
Destinatari principali	1
Obiettivi	1
Informazioni su questa serie di contenuti	2
Schemi degli agenti	3
Agenti di ragionamento di base	4
Architecture	4
Description	5
Funzionalità	6
Limitazioni	6
Casi di utilizzo comune	6
Guida all'implementazione	7
Riepilogo	7
Architecture	7
Description	8
Funzionalità	9
Casi di utilizzo comune	9
Guida all'implementazione	9
Riepilogo	10
Tool-based agenti per la chiamata di funzioni	10
Architecture	10
Description	11
Funzionalità	12
Casi di utilizzo comune	12
Guida all'implementazione	12
Riepilogo	13
Tool-based agenti per server	13
Architecture	13
Description	14
Funzionalità	15
Casi di utilizzo comune	15
Guida all'implementazione	15
Riepilogo	16
Computer-use agenti	16

Architecture	16
Description	17
Funzionalità	18
Casi di utilizzo comune	18
Guida all'implementazione	19
Riepilogo	19
Agenti di codifica	19
Architecture	19
Description	20
Funzionalità	21
Casi di utilizzo comune	21
Guida all'implementazione	22
Riepilogo	22
Agenti vocali e vocali	22
Architecture	22
Description	23
Funzionalità	24
Casi di utilizzo comune	24
Guida all'implementazione	25
Riepilogo	25
Agenti di orchestrazione del flusso di lavoro	25
Architecture	26
Description	26
Funzionalità	27
Casi di utilizzo comune	27
Guida all'implementazione	27
Riepilogo	28
Memory-augmented agents	28
Architecture	28
Description	29
Funzionalità	30
Casi di utilizzo comune	30
Implementazione di agenti con memoria aumentata	30
Implementazione dei prompt iniettati dalla memoria	31
Riepilogo	32
Agenti di simulazione e test	32

Architecture	32
Description	33
Funzionalità	34
Casi di utilizzo comune	34
Guida all'implementazione	35
Riepilogo	36
Agenti osservatori e di monitoraggio	36
Architecture	37
Description	37
Funzionalità	38
Casi di utilizzo comune	38
Guida all'implementazione	38
Riepilogo	39
Multi-agent collaborazione	39
Description	41
Funzionalità	42
Casi di utilizzo comune	42
Guida all'implementazione	42
Riepilogo	43
Conclusioni	43
Da asporto	44
Flussi di lavoro LLM	45
LLM-augmented Panoramica della cognizione	45
Flusso di lavoro per il concatenamento rapido	46
Description	47
Funzionalità	47
Casi di utilizzo comune	48
Flusso di lavoro per il routing	48
Funzionalità	49
Casi di utilizzo comune	49
Flusso di lavoro per la parallelizzazione	49
Funzionalità	51
Casi di utilizzo comune	51
Flusso di lavoro per l'orchestrazione	51
Funzionalità	53
Casi di utilizzo comune	53

Flusso di lavoro per valutatori e cicli Reflect-Refine	53
Casi di utilizzo comune	54
Funzionalità	54
Conclusioni	54
Modelli di flusso di lavoro agentici	56
Dai sistemi basati sugli eventi a quelli basati sulla cognizione aumentata	56
Event-driven architettura	56
Cognition-augmented flussi di lavoro	57
Approfondimenti fondamentali	59
Rapido concatenamento degli schemi della saga	59
Coreografia saga	59
Schema di concatenamento rapido	60
Coreografia dell'agente	60
Da asporto	63
Schemi di invio dinamici di routing	63
Invio dinamico	64
LLM-based routing	65
Router agente	66
Da asporto	67
Parallelizzazione e schemi di dispersione	67
Scatter-gather	68
LLM-based parallelizzazione (cognizione scatter-gather)	70
Parallelizzazione degli agenti	70
Da asporto	71
Schemi di orchestrazione Saga	71
Orchestrazione di eventi	72
Role-based sistema di agenti (orchestratore)	73
Supervisore	73
Da asporto	75
Evaluator Reflect-Refine Loop Patterns	75
Circuito di controllo del feedback	76
Circuito di controllo del feedback (valutatore)	78
Valutatore	78
Cose da asporto	79
Progettazione di flussi di lavoro agentici su AWS	79
Conclusioni	80

Cronologia dei documenti	82
.....	lxxxiii

Modelli e flussi di lavoro di intelligenza artificiale agentica su AWS

Aaron Sempf e Andrew Hooker, Amazon Web Services

Luglio 2025 ([cronologia dei documenti](#))

Le organizzazioni stanno adottando modelli linguistici di grandi dimensioni (LLMs) e agenti software per risolvere problemi dinamici e multidominio utilizzando una nuova disciplina architettonica chiamata *agentic pattern*. I pattern agentici sono modelli fondamentali e costrutti modulari utilizzati per progettare e orchestrare agenti di intelligenza artificiale orientati agli obiettivi in molti contesti.

Destinatari principali

Questa guida è destinata ad architetti, sviluppatori e leader di prodotto che desiderano creare applicazioni intelligenti che vadano oltre la logica statica, la logica simbolica e l'automazione deterministica.

Obiettivi

Questa guida fornisce un framework di progettazione e un approccio di implementazione per sistemi di agenti AI che operano in modo autonomo pur rimanendo controllabili e allineati agli obiettivi. Collega modelli architettonici basati sugli eventi con varie alternative agentiche, dimostrando come creare sistemi di agenti di livello di produzione utilizzando architetture native del cloud. In questa guida vengono trattati i seguenti argomenti:

- **Modelli di agenti:** i modelli di agente sono modelli di progettazione riutilizzabili che descrivono la struttura e il comportamento dei singoli agenti. Ciò include agenti di ragionamento, agenti di recupero potenziato, agenti di codifica, interfacce vocali, orchestratori di flussi di lavoro e sistemi collaborativi multiagente. Ogni modello illustra il modo in cui gli agenti percepiscono, ragionano, agiscono e apprendono, mappati. Servizi AWS
- **Flussi di lavoro LLM:** i flussi di lavoro si concentrano sul modo in cui gli agenti utilizzano il ragionamento. LLMs Esplorano le strategie di suggerimento e i meccanismi di pianificazione e descrivono come LLMs vengono utilizzati non solo per generare testo, ma anche per promuovere comportamenti strutturati, interpretabili e affidabili all'interno di un ciclo di agenti.

- Modelli di flusso di lavoro agentici: i modelli di flusso di lavoro descrivono come più agenti, strumenti e ambienti interagiscono per formare sistemi autonomi. Ciò include modelli per l'orchestrazione delle attività, la delega dei subagenti, il coordinamento basato sugli eventi, l'osservabilità e il controllo. Questi aspetti promuovono architetture AI scalabili, componibili e verificabili.

Informazioni su questa serie di contenuti

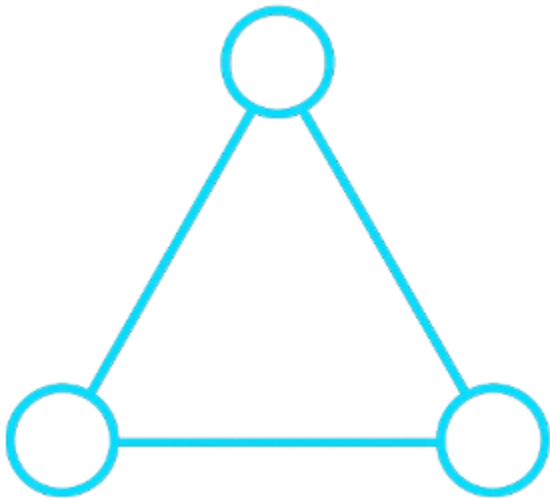
Questa guida fa parte di una serie sull'intelligenza artificiale agentica su AWS. Per ulteriori informazioni e per visualizzare le altre guide di questa serie, consulta [Agentic AI](#) sul sito Web Prescriptive Guidance. AWS

Schemi degli agenti

I pattern degli agenti sono elementi costitutivi riutilizzabili e componibili che possono essere adattati a domini, casi d'uso e livelli di complessità specifici. I sistemi Agentic differiscono, tuttavia, dalle applicazioni tradizionali. Alla base di tutti i progetti di agenti di intelligenza artificiale c'è un modello concettuale ancorato ai seguenti tre principi fondamentali:

- **Asincrono:** gli agenti operano in ambienti scarsamente accoppiati e ricchi di eventi
- **Autonomia:** gli agenti agiscono in modo indipendente, senza controllo umano o esterno
- **Agenzia:** gli agenti agiscono con uno scopo, per conto di un utente o di un sistema, verso obiettivi specifici

Il triangolo nel diagramma seguente rappresenta gli elementi costitutivi fondamentali di un agente software: percezione, ragione e azione. Ciò consente a un sistema agentic di osservare, prendere decisioni e agire all'interno del proprio ambiente.



In base alla progettazione, i modelli agentici forniscono un linguaggio di progettazione modulare per la creazione di sistemi di intelligenza artificiale, il che significa che sono accessibili, operativi, estensibili e pronti per la produzione. La progettazione di questi sistemi richiede un'attenzione particolare alle seguenti tre dimensioni correlate, che verranno ulteriormente discusse più avanti in questa guida.

In questa sezione

- [Agenti di ragionamento di base](#)

- [Tool-based agenti per la chiamata di funzioni](#)
- [Tool-based agenti per server](#)
- [Computer-use agenti](#)
- [Agenti di codifica](#)
- [Agenti vocali e vocali](#)
- [Agenti di orchestrazione del flusso di lavoro](#)
- [Memory-augmented agenti](#)
- [Agenti di simulazione e test](#)
- [Agenti osservatori e di monitoraggio](#)
- [Multi-agent collaborazione](#)

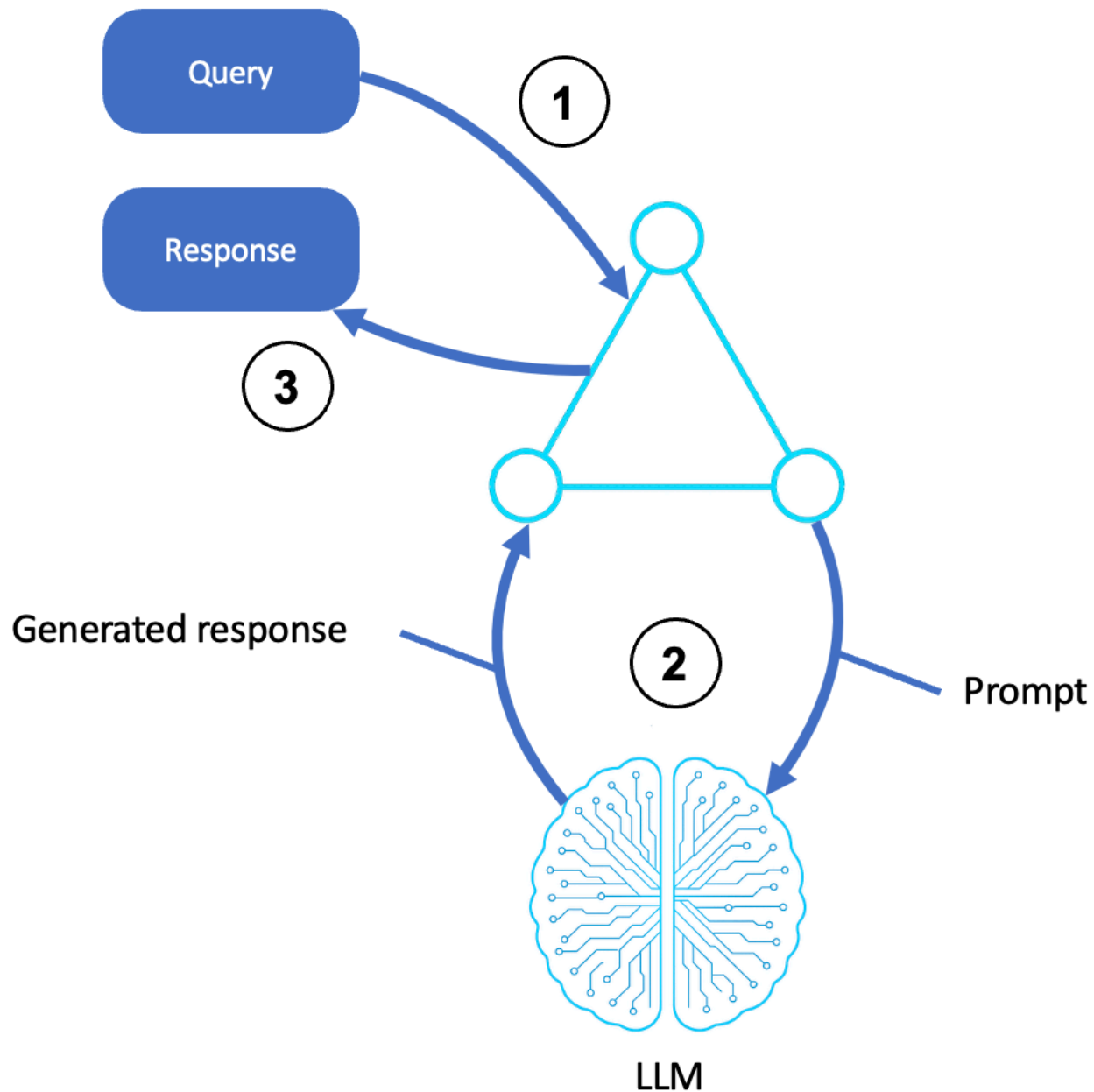
Agenti di ragionamento di base

Un agente di ragionamento di base è la forma più semplice di intelligenza artificiale agentica che esegue l'inferenza logica o il processo decisionale in risposta a una domanda. Accetta input da un utente o da un sistema ed elabora le domande e genera risposte utilizzando prompt strutturati.

Questo modello è utile per le attività che richiedono un ragionamento, una classificazione o un riepilogo in un unico passaggio in base a un determinato contesto. Non utilizza memoria, strumenti o gestione dello stato, il che lo rende privo di stato, leggero e altamente componibile in flussi di lavoro di grandi dimensioni.

Architecture

Il flusso di un agente di ragionamento di base è mostrato nel diagramma seguente:



Description

1. Riceve un input

- Un utente, un sistema o un agente upstream invia una richiesta o un'istruzione.
- L'input viene trasferito alla shell dell'agente o al livello di orchestrazione.
- Questo passaggio include qualsiasi preelaborazione, creazione di modelli rapidi e identificazione degli obiettivi.

2. Richiama l'LLM

- L'agente trasforma la query in un prompt strutturato e la invia a un LLM (ad esempio, tramite Amazon Bedrock).
- L'LLM genera una risposta basata sul prompt utilizzando conoscenze e contesto preformati.
- L'output generato può includere fasi di ragionamento (catena di pensiero), risposte finali o opzioni classificate.

3. Restituisce una risposta

- L'output generato viene inoltrato all'interfaccia dell'agente.
- Ciò può includere la formattazione, la postelaborazione o una risposta API.

Funzionalità

- Supporta il linguaggio naturale o l'input strutturato
- Utilizza la progettazione tempestiva per guidare il comportamento
- Senza stato e scalabile
- Può essere integrato in UI, CLI, API e pipeline

Limitazioni

- Nessuna memoria o consapevolezza storica
- Nessuna interazione con strumenti o fonti di dati esterne
- Limitato a ciò che il LLM conosce al momento dell'inferenza

Casi di utilizzo comune

- Domande e risposte conversazionali
- Spiegazioni e riassunti delle politiche
- Linee guida per prendere decisioni
- Flussi di chatbot leggeri e automatizzati
- Classificazione, etichettatura e punteggio

Guida all'implementazione

È possibile utilizzare i seguenti strumenti e servizi per creare un agente di ragionamento di base:

- Amazon Bedrock per invocazioni LLM (Anthropic, AI21, Meta)
- Amazon API Gateway o AWS Lambda esporlo come microservizio stateless
- Modelli di prompt archiviati in Parameter Store o come AWS Secrets Manager codice

Riepilogo

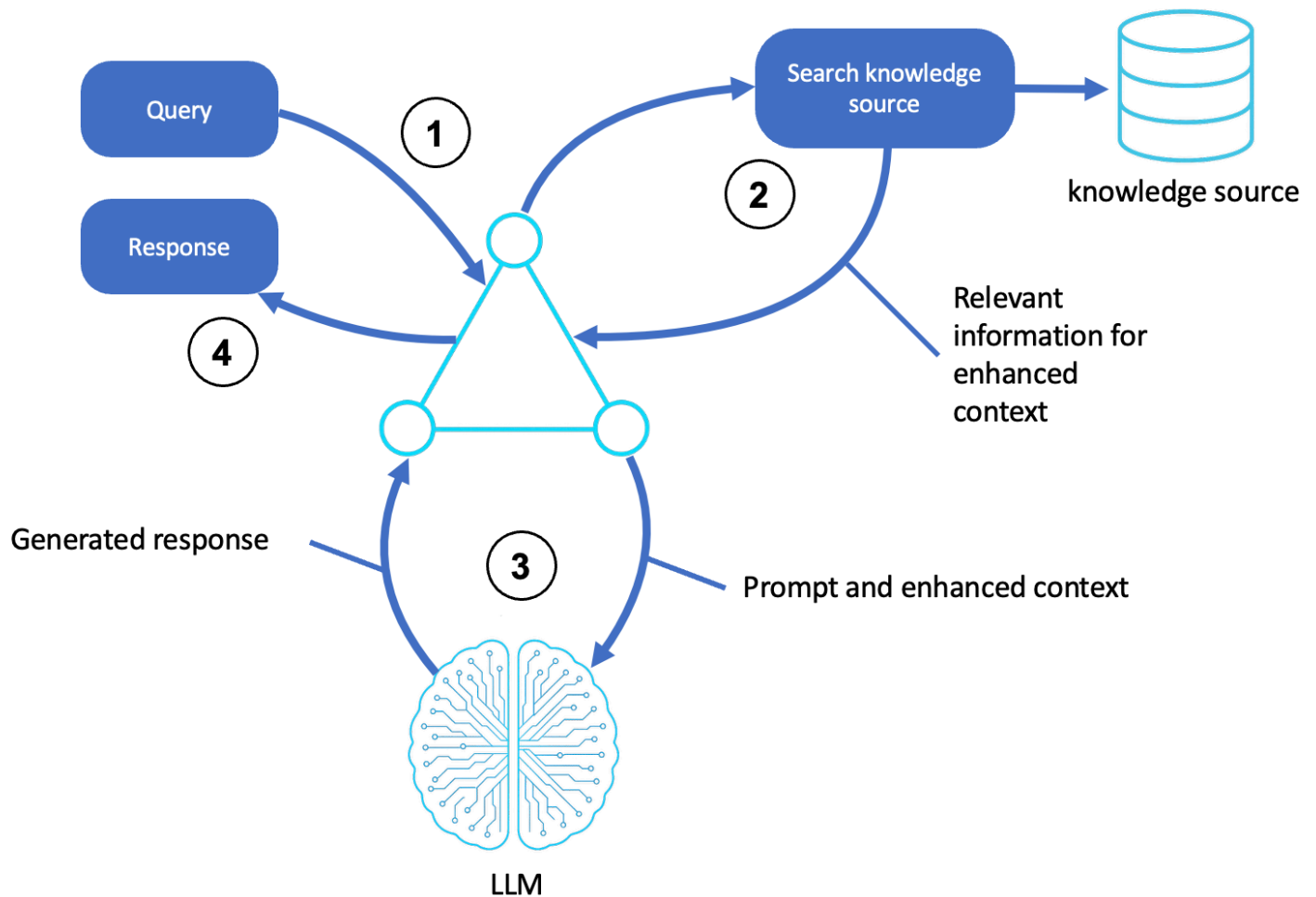
L'agente di ragionamento di base è fondamentale grazie alla sua struttura semplice. Ha funzionalità fondamentali che trasformano gli obiettivi in percorsi di ragionamento che portano a risultati intelligenti. Questo modello è spesso un punto di partenza per modelli avanzati, come agenti basati su strumenti e agenti che utilizzano la generazione aumentata di recupero (RAG). È anche un componente affidabile e modulare di flussi di lavoro di grandi dimensioni.

Agente RAG

Retrieval-augmented generazione (RAG) è una tecnica che combina il recupero delle informazioni con la generazione di testo per creare risposte accurate e contestuali. RAG consente agli agenti di recuperare informazioni esterne pertinenti prima di avviare il LLM. Estende la memoria effettiva e l'accuratezza del ragionamento di un agente basando le sue decisioni su informazioni aggiornate, fattuali o specifiche del dominio. A differenza dei LLM stateless che si basano esclusivamente su pesi predefiniti, RAG dispone di un livello di ricerca delle conoscenze esterno che migliora dinamicamente i prompt in base al contesto.

Architecture

La logica del pattern RAG è illustrata nel diagramma seguente:



Description

1. Riceve una richiesta

- Un utente o un sistema upstream invia una richiesta o un obiettivo all'agente.
- La shell dell'agente accetta la richiesta e la formatta come richiesta di ragionamento.

2. Cerca una fonte esterna

- L'agente identifica concetti e intenti a partire dall'interrogazione.
- Interroga una fonte di conoscenza, ad esempio un archivio vettoriale, un database o un indice di documenti utilizzando la ricerca semantica o la corrispondenza di parole chiave.
- I passaggi, i documenti o le entità più rilevanti vengono recuperati per essere utilizzati nella fase successiva.

3. Genera una risposta contestuale

- L'agente aumenta il prompt con le informazioni recuperate, formando un input contestuale per l'LLM.
- L'LLM elabora qualsiasi input utilizzando il ragionamento generativo (ad esempio, catena di pensiero o riflessione) per produrre una risposta accurata.

4. Restituisce l'output finale

- L'agente prepara l'output inserendolo in qualsiasi intestazione di comunicazione o nella formattazione richiesta, quindi lo restituisce all'utente o al sistema chiamante.
- (Facoltativo) I documenti recuperati e l'output LLM possono essere registrati, valutati e archiviati in memoria per future interrogazioni.

Funzionalità

- Fact-grounded output anche in domini a coda lunga o specifici dell'azienda
- Estensione della memoria senza ottimizzazione del modello
- Contesto dinamico basato su ogni query e stato dell'utente
- Pienamente compatibile con database vettoriali, indici semantici e filtraggio dei metadati

Casi di utilizzo comune

- Assistenti alla conoscenza aziendale
- Bot per la conformità normativa
- Copiloti dell'assistenza clienti
- Search-enhanced chatbot
- Agenti di documentazione per sviluppatori

Guida all'implementazione

Utilizzate i seguenti strumenti e servizi per creare un agente che utilizzi RAG:

- Amazon Bedrock per invocazioni LLM
- Amazon Kendra OpenSearch o Amazon Aurora per la documentazione o una ricerca di dati strutturati

- Amazon Simple Storage Service (Amazon S3) Simple Storage Service (Amazon S3) per l'archiviazione di documenti
- AWS Lambda per orchestrare la ricerca, i prompt e l'inferenza LLM
- Knowledge-based integrazioni con agenti (utilizzando plug-in di memoria, retriever semantici o Amazon Bedrock)

Riepilogo

Agent RAG collega il ragionamento basato su modelli statici all'intelligenza dinamica del mondo reale. Fornisce agli agenti la capacità di cercare ciò che non sanno, sintetizzare le risposte a partire dalle conoscenze acquisite e produrre risposte verificabili e altamente affidabili.

I pattern RAG sono la base per la creazione di agenti intelligenti che scalano l'accesso alla conoscenza senza riqualificazione. Spesso è un precursore di schemi di orchestrazione più complessi che coinvolgono l'uso di strumenti, la pianificazione e la memoria a lungo termine.

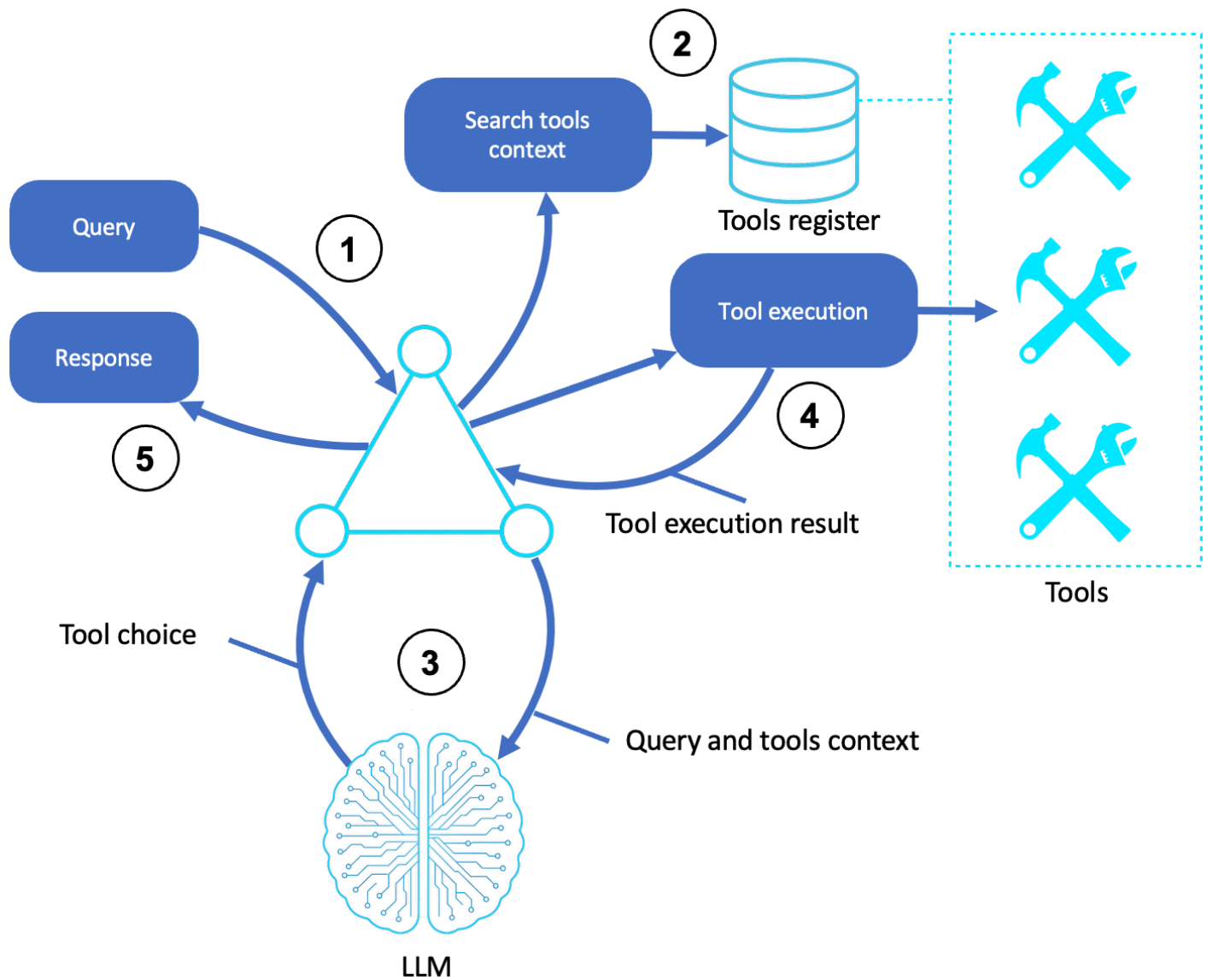
Tool-based agenti per la chiamata di funzioni

Tool-based gli agenti estendono le capacità degli agenti di ragionamento richiamando funzioni o API esterne per completare attività che vanno oltre il semplice ragionamento linguistico. Questo modello utilizza un LLM per decidere quale strumento utilizzare, quindi genera argomenti di chiamata e incorpora l'output di uno strumento nel suo ciclo di ragionamento.

Questo modello consente agli agenti di agire anziché limitarsi a fornire risposte. L'interfaccia dello strumento rappresenta qualsiasi funzionalità richiamabile, dai calcoli aritmetici e dalle ricerche nei database alle API esterne e ai servizi cloud.

Architecture

Nel diagramma seguente è illustrato un agente basato su strumenti per la chiamata di funzioni:



Description

1. Riceve una richiesta

- L'agente riceve una richiesta o un'attività in linguaggio naturale dall'utente o dal sistema chiamante.

2. Cerca strumenti

- L'agente utilizza metadati interni o un registro degli strumenti per cercare gli strumenti, gli schemi e le funzionalità pertinenti disponibili.

3. Seleziona e richiama gli strumenti

- L'LLM riceve i metadati della query e dello strumento (ad esempio, nomi delle funzioni, tipi di input e descrizioni) nel relativo prompt.
 - Sceglie lo strumento più pertinente, costruisce argomenti di input e restituisce una chiamata di funzione strutturata.
4. Esegue lo strumento scelto
- La shell dell'agente o il tool runner esegue la funzione selezionata e restituisce il risultato (ad esempio, un output API, un valore del database o un calcolo).
5. Restituisce una risposta
- L'LLM trasmette i risultati all'agente, direttamente o come parte di un prompt aggiornato. Quindi restituisce un risultato in linguaggio naturale.

Funzionalità

- Selezione dinamica degli utensili in base al contesto dell'attività
- Schema-based prompting (OpenAPI, schema AWS JSON, interfaccia di funzione)
- Interpretazione dei risultati e concatenamento degli output nel ragionamento
- Operazioni senza stato o con riconoscimento della sessione

Casi di utilizzo comune

- Assistenti virtuali con accesso esterno ai dati
- Calcolatori e stimatori finanziari
- API-based lavoratori della conoscenza
- LLM che richiamano, SageMaker endpoint AWS Lambda Amazon e servizi SaaS

Guida all'implementazione

Usa quanto segue per creare agenti basati su strumenti per chiamare le funzioni:

- Amazon Bedrock con supporto per le chiamate di funzioni (Anthropic Claude)
- AWS Lambda come backend per l'esecuzione degli strumenti
- Amazon API Gateway o AWS Step Functions per l'orchestrazione degli strumenti

- Amazon DynamoDB o Amazon Relational Database Service (Amazon RDS) per i metadati degli strumenti sensibili al contesto
- Amazon EventBridge Pipeline o AWS Step Functions che mappano gli stati per indirizzare gli output

Riepilogo

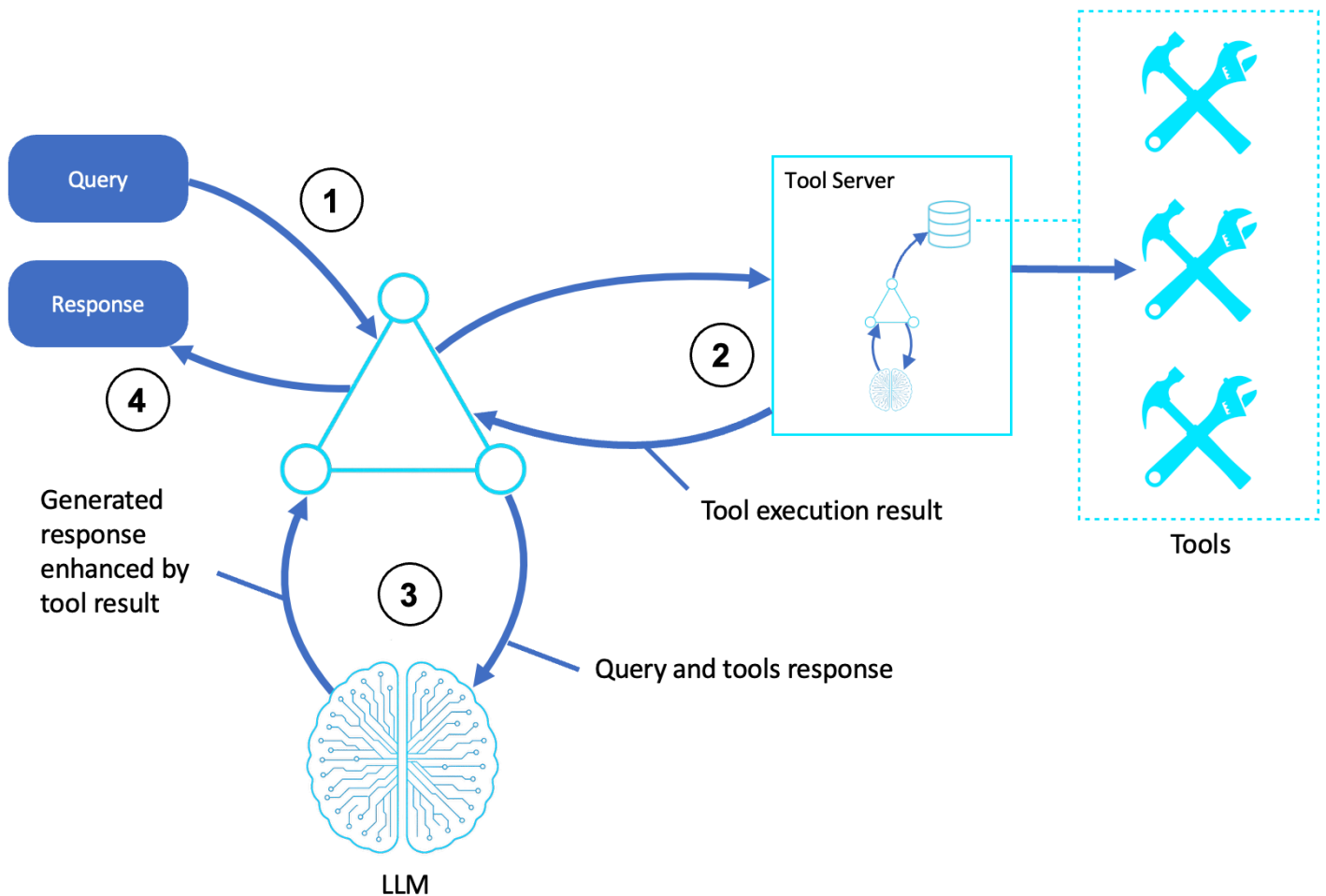
Tool-based Gli agenti che chiamano funzioni rappresentano il passaggio dalla comprensione del linguaggio all'esecuzione di azioni. Questi agenti richiamano strumenti dinamici e sensibili al contesto pur mantenendo il ragionamento LLM, trasformando gli assistenti passivi in sistemi che completano le attività, accedono ai servizi e integrano le operazioni aziendali. Questo modello è una componente importante dell'intelligenza artificiale agentica in ambito aziendale, specialmente se combinato con schemi dichiarativi, framework di autorizzazione e sistemi multiagente.

Tool-based agenti per server

Tool-based gli agenti per server migliorano gli agenti che richiamano le funzioni delegando l'esecuzione degli strumenti a un server esterno dotato di un ambiente di runtime dedicato per strumenti, script e agenti compositi. A differenza delle chiamate di funzioni in linea che l'agent loop seleziona e richiama, gli agenti basati su server esternalizzano la logica e la pipeline di esecuzione ad altri agenti o sistemi. Ciò fornisce funzionalità avanzate come il concatenamento di più strumenti, l'esecuzione isolata e il ragionamento specializzato. I tool server sono ideali per azioni complesse, basate sullo stato o che richiedono molte risorse, in cui gli strumenti stessi possono comportare modelli di intelligenza artificiale, regole aziendali o ambienti separati.

Architecture

Di seguito è riportato uno schema per agenti basati su strumenti per server:



Description

1. Riceve una richiesta

- Un utente o un sistema invia una richiesta alla shell dell'agente.
- L'agente interpreta la query e si prepara a inviarla a un server di strumenti.

2. Esegue i processi del server degli strumenti

- L'agente invia l'operazione, insieme ai parametri strutturati, a un server di strumenti.
- Il tool server può quindi:
 - Eseguire script o logica in sistemi di elaborazione dedicati (ad esempio AWS Lambda, container o Amazon SageMaker)
 - Utilizzare il proprio subagente con il ragionamento LLM per selezionare ed eseguire gli strumenti
 - Gestire dipendenze, nuovi tentativi o flussi di esecuzione in più fasi
 - Inviare i risultati all'agente principale quando l'attività è completa

3. Utilizza il ragionamento LLM con l'output dello strumento

- L'agente richiama un LLM, passando la query originale e il risultato del toolserver come parte del prompt.
- L'LLM sintetizza una risposta che incorpora le informazioni appena acquisite.

4. Restituisce una risposta

- L'agente restituisce una risposta in linguaggio naturale o strutturata all'utente o al sistema chiamante.
- (Facoltativo) I risultati possono essere archiviati in memoria o nei registri di controllo.

Funzionalità

- Gli strumenti vengono richiamati al di fuori del ciclo di esecuzione dell'agente principale
- L'esecuzione degli strumenti può coinvolgere chiamate LLM, catene logiche o subagenti
- L'agente funge da controller o dispatcher, non solo da strumento wrapper
- Consente la componibilità, la scalabilità e l'isolamento della logica

Casi di utilizzo comune

- Orchestrazione di catene di modelli (ad esempio, combinando LLM, visione e codice)
- AI-driven condotte di automazione
- DevOps agenti assistenti con script runner
- Agenti complessi di calcolo, simulazione o ottimizzazione finanziaria
- Strumenti multimodali (ad esempio, combinando audio, documentazione e azione)

Guida all'implementazione

È possibile creare questo modello utilizzando quanto segue: Servizi AWS

- Amazon Bedrock (host dell'agente e inferenza LLM)
- AWS Lambda, Amazon ECS o Amazon SageMaker endpoint come runtime del server degli strumenti AWS Fargate
- Amazon API Gateway o AWS App Runner per esporre le API dei server degli strumenti

- Amazon EventBridge per la messaggistica disaccoppiata agente-strumento
- AWS Step Functions o AWS AppFabric per comporre una logica multiagente sul server degli strumenti

Riepilogo

Tool-based gli agenti che utilizzano server sono altamente modulari e scalabili. Separano la logica decisionale dall'esecuzione, il che consente all'agente principale di rimanere leggero mentre trasferisce le azioni complesse o sensibili su altri sistemi. Questo è importante per l'intelligenza artificiale agentica di livello aziendale, specialmente in ambienti che richiedono governance, osservabilità, isolamento, composizione dinamica o qualsiasi combinazione di questi elementi.

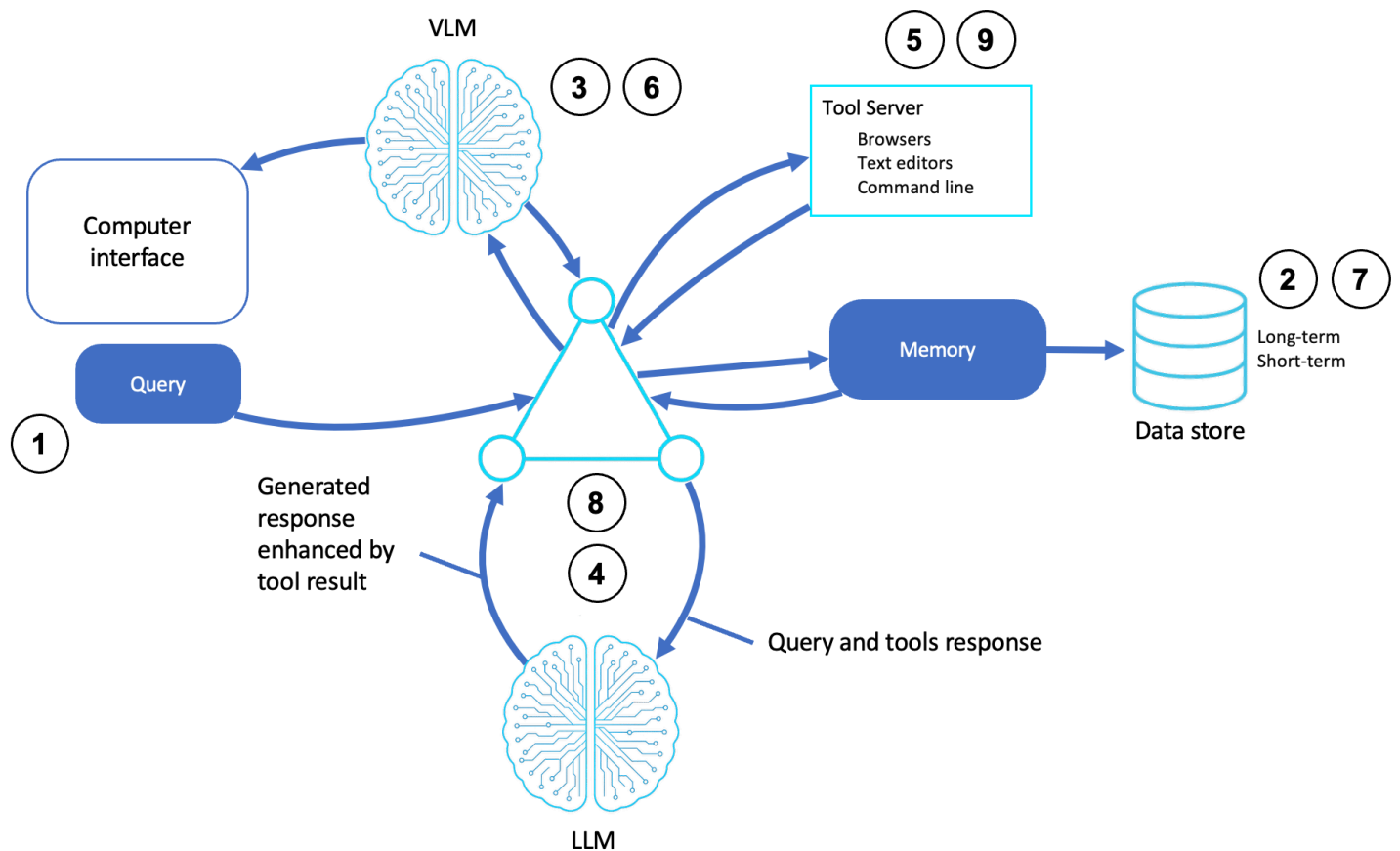
Computer-use agenti

Computer-use gli agenti possono simulare o controllare ambienti digitali come browser, terminali, file system e applicazioni. Questi agenti interpretano l'intento dell'utente, interagiscono con interfacce visive e testuali ed eseguono azioni mirate combinando ragionamenti LLM, modelli di linguaggio visivo (VLM) e server di strumenti che eseguono comandi o simulano eventi di input.

Questo modello è importante per le automazioni di intelligenza artificiale pratiche, in cui l'agente non funziona solo come assistente ma anche come proxy che esegue azioni come farebbe un essere umano, spesso utilizzando gli stessi strumenti e ambienti.

Architecture

Nel diagramma seguente è illustrato un modello di agente utilizzato dal computer:



Description

1. Riceve una richiesta

- Un'attività o una richiesta viene fornita tramite un'interfaccia utente, un'API o un'interfaccia in linguaggio naturale.

2. Accede alla memoria

- L'agente recupera la memoria a breve e lungo termine per richiamare comandi, obiettivi e stati del sistema passati.

3. Analizza il contesto visivo

- Un VLM osserva lo schermo del computer, lo stato del sistema o gli elementi dell'interfaccia utente per comprendere un determinato contesto e identificare gli elementi utilizzabili.

4. Motivi tramite un LLM

- L'LLM combina la query, lo stato della memoria, lo strumento e la risposta del server per determinare l'azione successiva.

5. Interagisce con il server degli strumenti

- L'agente richiama strumenti ospitati su un server, che possono includere quanto segue:
 - Browser (ad esempio, Chrome headless) e ambienti shell
 - Editor di testo e codice
 - Interfacce di script personalizzate

6. Aggiorna gli input visivi

- Se l'interfaccia utente del sistema cambia o sono necessarie ulteriori osservazioni, il VLM può rianalizzare lo stato dello schermo o i buffer di testo.

7. Aggiorna la memoria

- Le nuove informazioni, gli stati del sistema o il feedback degli utenti vengono scritti nella memoria a breve e lungo termine.

8. Formula decisioni e spiegazioni finali

- L'LLM sintetizza i risultati o consiglia azioni in base all'output dell'interrogazione e dello strumento.

9. Restituisce una risposta

- L'agente restituisce i risultati all'interfaccia (ad esempio, un'attività completata, una conferma o un contenuto generato).

Funzionalità

- Ragionamento multimodale con input visivi e testuali
- Controllo delle applicazioni tramite input o simulati API-driven
- Gestione della memoria per uno stato persistente
- Autonomia nell'esecuzione della sequenza (flussi a più fasi)

Casi di utilizzo comune

- Sviluppatori di intelligenza artificiale che scrivono ed eseguono codice negli IDE
- Computer-use agenti per flussi di lavoro digitali ripetitivi
- Utenti simulati per il test del software e il controllo della qualità
- Agenti di accessibilità per navigare nelle interfacce utente tramite istruzioni vocali o di alto livello
- Automazione robotica intelligente dei processi (RPA) potenziata dal ragionamento

Guida all'implementazione

- È possibile creare questo modello utilizzando quanto segue: Servizi AWS
- Amazon Bedrock per la LLM-based pianificazione e il ragionamento
- Amazon Elastic Compute Cloud (Amazon EC2) o notebook SageMaker Amazon per eseguire AWS Lambda server di strumenti con ambienti di interfaccia utente simulati
- Amazon Simple Storage Service (Amazon S3) Simple Storage Service (Amazon S3) o Amazon DynamoDB per la persistenza della memoria
- Amazon Rekognition (o modelli personalizzati) per l'analisi delle immagini dell'interfaccia utente in scenari ibridi
- Amazon CloudWatch Logs o AWS X-Ray per l'osservabilità e gli audit trail

Riepilogo

Computer-use gli agenti agiscono come operatori digitali autonomi, colmando il divario tra interazioni e azioni uomo-computer. AI-driven Incorporando memoria, orchestrazione degli strumenti e VLM, questi agenti possono interagire in modo adattivo con i sistemi progettati per gli esseri umani, eseguire azioni, aggiornare file, navigare nei menu e generare risposte.

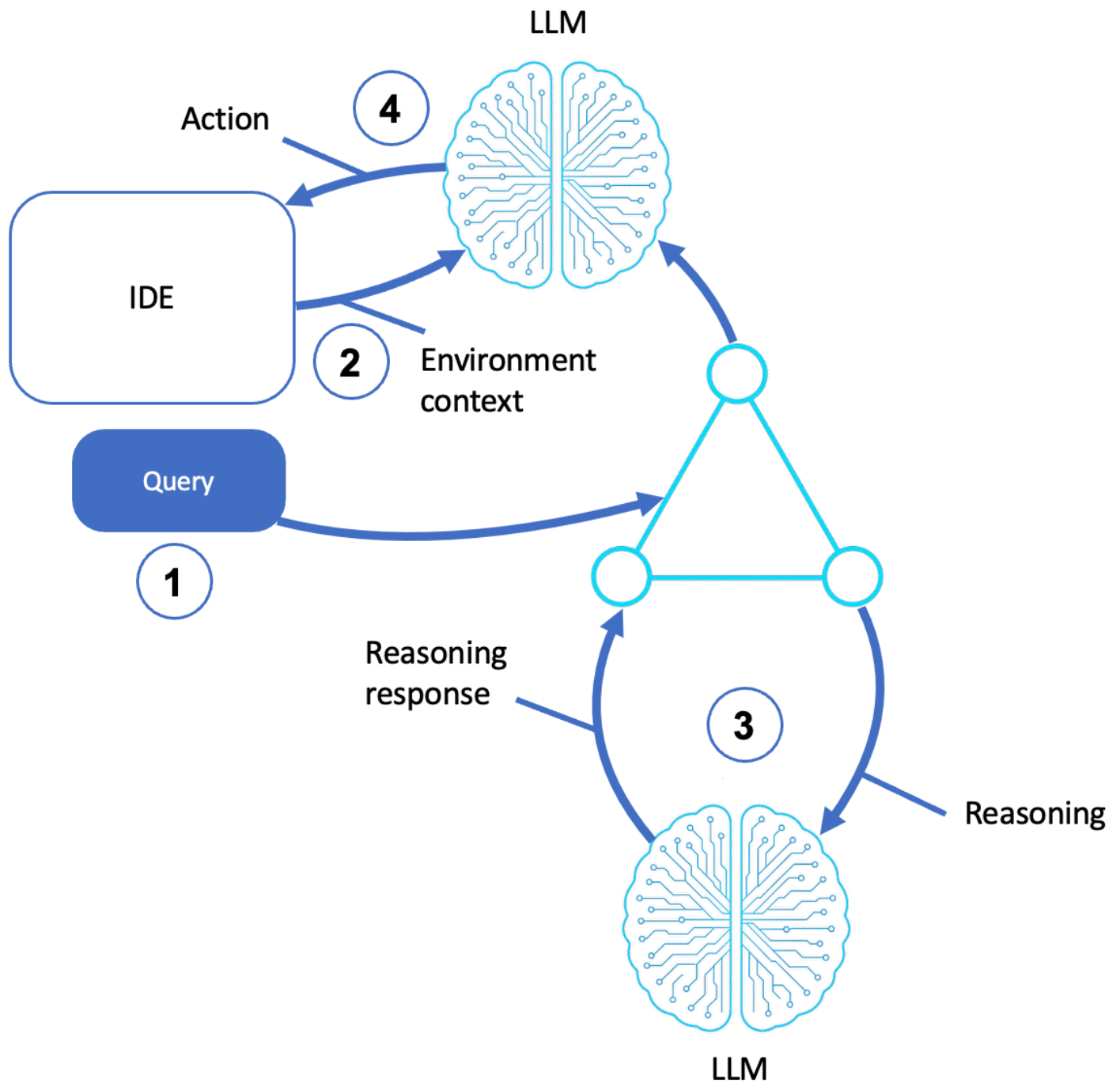
Agenti di codifica

Gli agenti di codifica possono ragionare sulle attività di programmazione, generare o modificare codice e interagire con ambienti di sviluppo, come IDE e CLI. Questi agenti combinano la comprensione del linguaggio naturale con il ragionamento strutturato per assistere, potenziare e automatizzare lo sviluppo del software, dalla generazione di funzioni alla correzione di bug e alla creazione di test.

A differenza degli strumenti di completamento automatico, gli agenti di codifica interpretano attivamente gli obiettivi degli utenti, interrogano il contesto nell'ambiente di sviluppo (ad esempio, apre i file e rileva gli errori), identificano i requisiti e quindi propongono ed eseguono azioni.

Architecture

Un modello di agente di codifica è illustrato nel diagramma seguente:



Description

1. Riceve una richiesta

- L'utente fornisce istruzioni in linguaggio naturale tramite una palette di comandi, una finestra di chat o una CLI (ad esempio, «Aggiungi registrazione a questa funzione» o «Refactor for readability»).
2. Estrae il contesto dell'ambiente
 - L'agente raccoglie il contesto dall'IDE, inclusi file attivi, posizione del cursore, frammenti di codice e tabelle dei simboli.
 - Emette messaggi di errore, risultati dei test e output di altri agenti.
 3. Ragionamento LLM
 - L'agente invia un prompt, incluso l'interrogazione e il contesto ambientale, a un LLM.
 - L'LLM esegue un ragionamento per determinare quanto segue:
 - Cosa deve cambiare
 - Come generare una soluzione
 - Qualsiasi fase di rifattorizzazione, riscrittura o codifica
 4. Esegue azioni
 - L'LLM restituisce l'output all'agente e lo importa nell'ambiente IDE o di runtime.
 - Ciò può includere l'inserimento o la modifica di codice, la generazione di commenti o documentazione e l'attivazione di attività successive di compilazione, test e linting.

Funzionalità

- High-contextual consapevolezza (ad esempio, stato IDE, cursore e albero della sintassi)
- Ragionamento iterativo degli obiettivi e del feedback
- Pianificazione opzionale del codice e separazione delle azioni (ad esempio, prima ragionare e poi agire)
- Funziona in flussi di lavoro di sviluppo sincroni o asincroni

Casi di utilizzo comune

- Generazione di codice dalle descrizioni delle attività
- Rifattorizzazione e ottimizzazione del codice
- Test-case generazione e convalida
- Spiegazione degli errori e debug

- Assistenti alla documentazione
- Copiloti di programmazione accoppiati

Guida all'implementazione

- È possibile creare questo modello utilizzando i seguenti strumenti e: Servizi AWS
- Amazon Bedrock per la LLM-driven generazione e il ragionamento
- Amazon Q Developer per suggerimenti e completamenti di codifica
- AWS Lambda o Amazon Elastic Container Service (Amazon ECS) per l'esecuzione e il test di ambienti sandbox
- AWS Cloud9, estensioni VS Code o integrazioni IDE personalizzate per ospitare e valutare il contesto
- Amazon Simple Storage Service (Amazon S3) Simple Storage Service (Amazon S3) per l'archiviazione di richieste, risposte e cronologia delle revisioni intermedie

Riepilogo

Gli agenti di codifica sono nuovi strumenti di AI-powered sviluppo in grado di interpretare il linguaggio naturale, analizzare il contesto, generare modifiche al codice in più fasi e integrarsi con il ciclo di vita dello sviluppo del software.

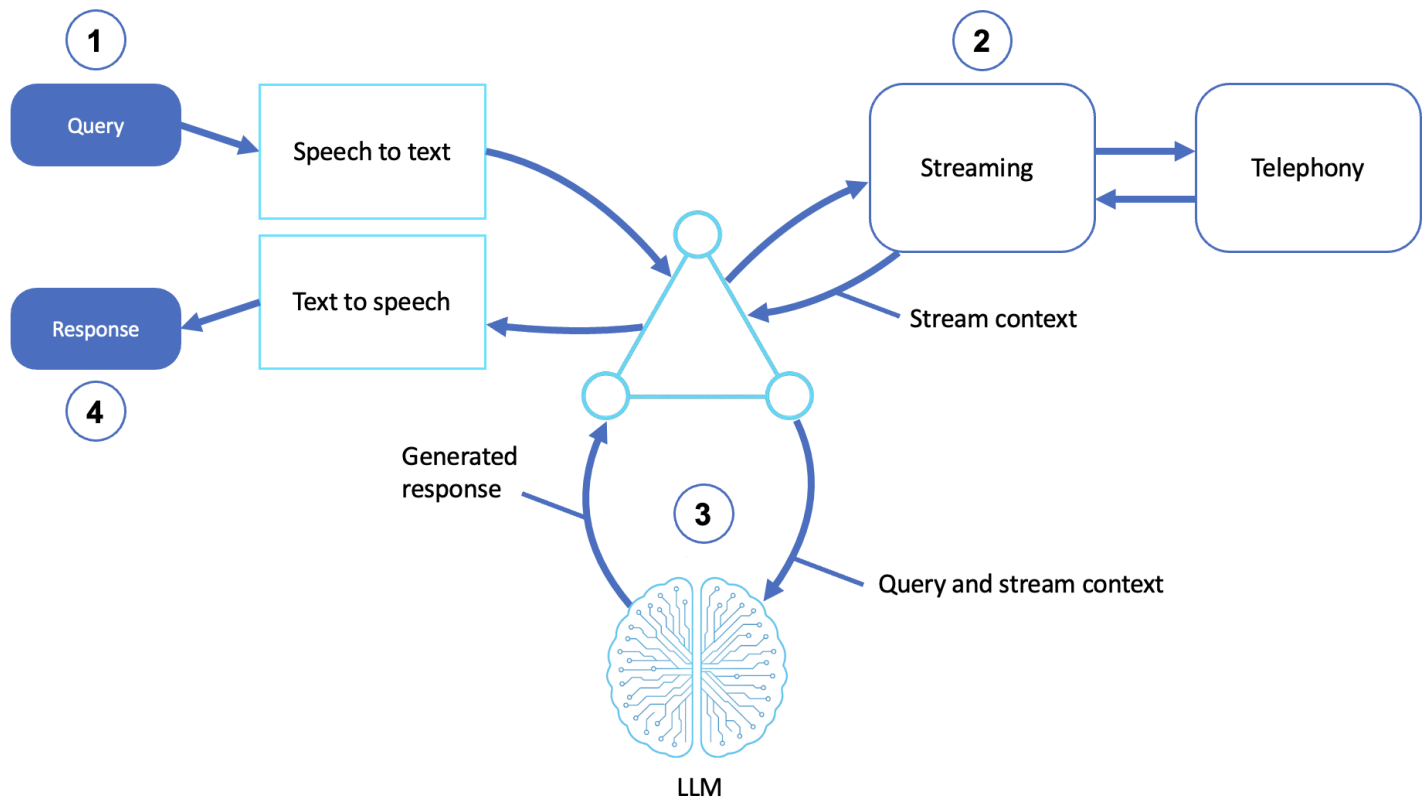
Agenti vocali e vocali

Gli agenti vocali e vocali interagiscono con gli utenti attraverso il dialogo vocale. Questi agenti integrano il riconoscimento vocale, la comprensione del linguaggio naturale e la sintesi vocale per consentire l'intelligenza artificiale conversazionale su telefonia, dispositivi mobili, web e piattaforme integrate.

Gli agenti vocali sono particolarmente efficaci in ambienti a mani libere, in tempo reale o basati sull'accessibilità. Combinando interfacce di streaming con il LLM-powered ragionamento, facilitano interazioni ricche e dinamiche che risultano naturali per gli utenti.

Architecture

Un agente vocale e vocale è illustrato nel diagramma seguente:



Description

1. Riceve una richiesta vocale

- L'utente invia una richiesta a un telefono, un microfono o un sistema integrato.
- Un modulo speech-to-text (STT) converte l'audio in testo.

2. Integra il contesto di streaming e telefonia

- L'agente utilizza un'interfaccia di streaming per gestire l'audio I/O in tempo reale.
- Se viene implementata in un contact center o in un contesto di telecomunicazioni, l'integrazione della telefonia gestisce il routing delle sessioni, l'input multifrequenza a due toni (DTMF) e il trasporto multimediale.

Nota: DTMF si riferisce ai toni generati quando si premono i pulsanti sulla tastiera di un telefono. Nel contesto dell'integrazione del contesto di streaming e telefonia negli agenti vocali, il DTMF viene utilizzato come meccanismo di immissione del segnale durante una telefonata, in particolare nei sistemi di risposta vocale interattiva (IVR). Gli ingressi DTMF consentono all'agente di:

- Riconoscere le selezioni del menu (ad esempio, «Premi 1» per la fatturazione). Premi 2 per ricevere assistenza.»)
- Raccogli input numerici (ad esempio numeri di conto, PIN e numeri di conferma)
- Attiva flussi di lavoro o transizioni di stato nei flussi di chiamata
- Passa dalla voce al tono tattile quando necessario

1. Motivi tramite il contesto del flusso LLM

- La query viene inviata all'agente, che la trasmette, insieme a tutti i metadati della sessione (ad esempio, ID chiamante, contesto precedente), a un LLM.
- L'LLM genera una risposta, possibilmente utilizzando una strategia di catena di pensiero o una memoria a più turni se l'interazione è in corso.

2. Restituisce una risposta vocale

- L'agente converte la sua risposta in voce utilizzando la sintesi vocale (TTS).
- Restituisce l'audio all'utente tramite un canale vocale.

Funzionalità

- Real-time comprensione e generazione del parlato
- Multilingue I/O con supporto STT e TTS
- Integrazione con API di telefonia o streaming
- Consapevolezza della sessione e trasferimento della memoria tra un turno e l'altro

Casi di utilizzo comune

- Sistemi IVR conversazionali
- Addetti alla reception e programmatori di appuntamenti virtuali
- Voice-driven agenti dell'helpdesk
- assistenti vocali indossabili
- Interfacce vocali per case intelligenti e strumenti di accessibilità

Guida all'implementazione

È possibile creare questo modello utilizzando i seguenti strumenti e: Servizi AWS

- Amazon Lex V2 o Amazon Transcribe per STT
- Amazon Polly per TTS
- Amazon Chime SDK, Amazon Connect Customer o Amazon Interactive Video Service (Amazon IVS) per streaming e telefonia
- Amazon Bedrock per ragionare con Anthropic, AI21 o altri modelli di base
- AWS Lambda per connettere STT, LLM, TTS e il contesto della sessione

(Facoltativo) I miglioramenti aggiuntivi possono includere quanto segue:

- Amazon Kendra OpenSearch o per RAG sensibile al contesto
- Amazon DynamoDB per la memoria di sessione
- Amazon CloudWatch Logs e AWS X-Ray per la tracciabilità

Riepilogo

Gli agenti vocali e vocali sono sistemi intelligenti che interagiscono attraverso conversazioni naturali. Integrando le interfacce vocali con il ragionamento LLM e l'infrastruttura di streaming in tempo reale, gli agenti vocali consentono interazioni fluide, accessibili e scalabili.

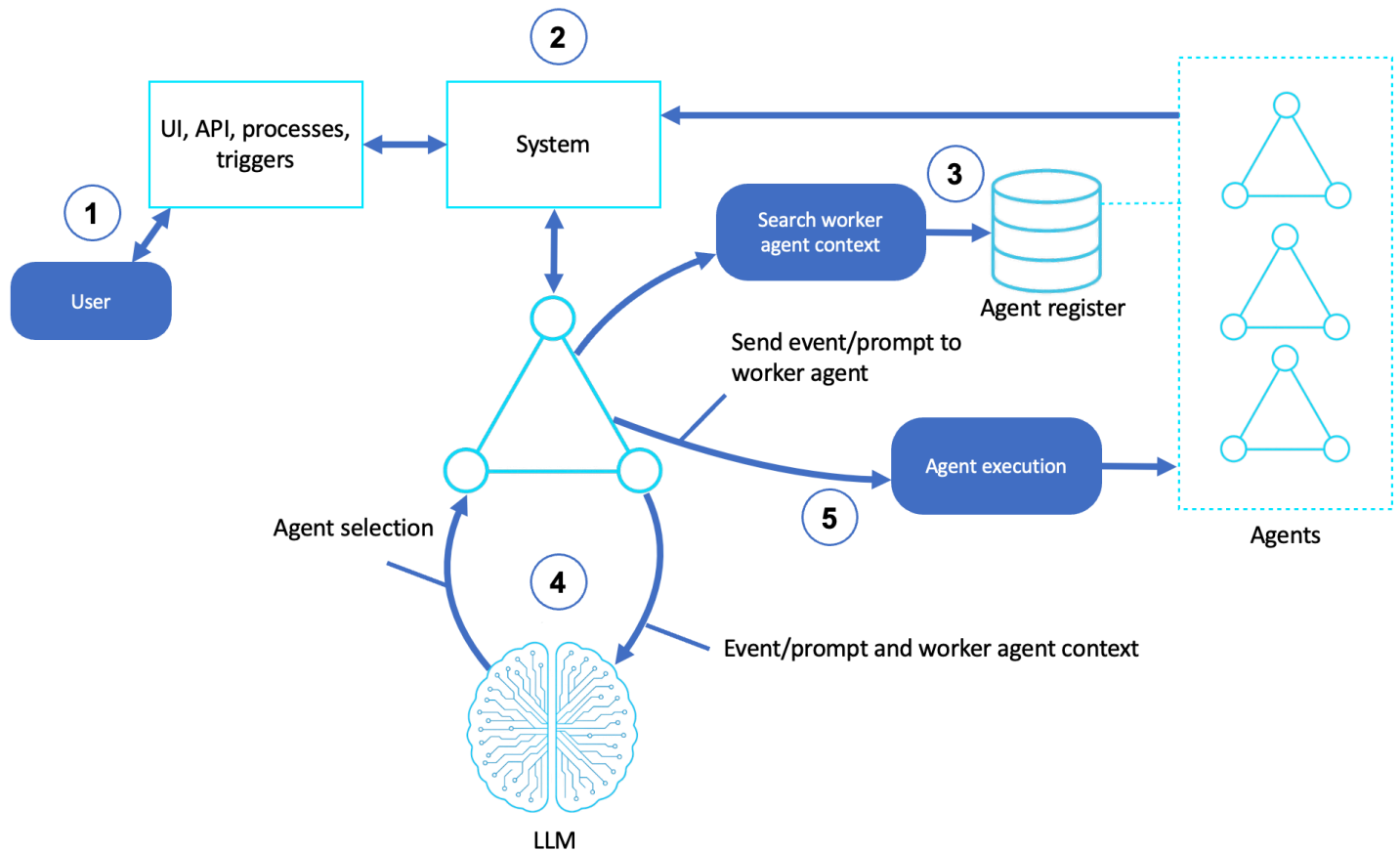
Agenti di orchestrazione del flusso di lavoro

Gli agenti di orchestrazione del flusso di lavoro gestiscono e coordinano attività, processi e servizi in più fasi su sistemi distribuiti. Piuttosto che ragionare e agire isolatamente, questi agenti delegano il lavoro a subagenti o altri sistemi, mantengono il contesto di esecuzione e si adattano in base a risultati intermedi.

Questi agenti sono una parte fondamentale dei flussi di automazione. Sono particolarmente utili quando si gestiscono attività di lunga durata, composizioni multiagente e integrazioni tra domini in cui è necessario chiamare vari agenti e strumenti in sequenza o in modo condizionale.

Architecture

Un agente di orchestrazione del flusso di lavoro è illustrato nel diagramma seguente:



Description

1. Riceve l'input dell'utente

- Un utente (o un trigger esterno) avvia un'attività tramite un'interfaccia utente, un'API o un evento di sistema.

2. Gestisce gli eventi di sistema

- Un componente di sistema riceve la richiesta ed emette un evento o un comando che richiede l'orchestrazione.

3. Recupera il contesto

- L'agente di workflow interroga le knowledge base e i registri degli agenti per trovare l'agente di lavoro giusto per l'attività in base a metadati, dominio e percentuale di successo precedente.

4. Seleziona un agente LLM

- Un LLM aiuta a selezionare l'agente o il piano di flusso di lavoro migliore analizzando la descrizione delle attività e le opzioni disponibili.
- Può anche formulare istruzioni specifiche per l'attività da inviare a un agente selezionato.

5. Delega ed esegue

- L'agente di lavoro scelto riceve l'evento o il prompt e inizia a eseguire i comandi.
- Può tenere traccia dello stato di esecuzione, riprovare in caso di errore e passare i risultati intermedi all'agente successivo della sequenza.

Funzionalità

- Composizione degli agenti (ad esempio supervisori, agenti collaboratori e strumenti)
- Event-driven o esecuzione programmata
- Monitoraggio della memoria e dello stato nel tempo
- Orchestrazione gerarchica o parallela delle attività (sincrona rispetto ai flussi di lavoro asincroni)
- Selezione e concatenamento dinamici degli agenti

Casi di utilizzo comune

- Automazione in più fasi (ad esempio, inserimento e creazione di report di dati)
- Instradamento e intensificazione dell'assistenza clienti (ad esempio, agent-as-coordinator)
- Gli agenti di intelligenza artificiale si coordinano con umani e bot all'interno dello stesso ciclo
- Automatizza i processi aziendali utilizzando la logica LLM-powered
- I sistemi ibridi combinano agenti di intelligenza artificiale e strumenti di orchestrazione tradizionali

Guida all'implementazione

È possibile creare questo modello utilizzando i seguenti strumenti e: Servizi AWS

- Amazon Bedrock per il ragionamento e la selezione degli agenti
- AWS Step Functions o Amazon EventBridge per la composizione del flusso di lavoro
- AWS Lambda come unità di esecuzione o task runner

- Amazon DynamoDB, Amazon Simple Storage Service (Amazon S3) o Amazon RDS per tenere traccia di stati e risultati
- AWS AppFabric o Amazon AppFlow per il coordinamento tra sistemi
- (Facoltativo) Usa Amazon SageMaker run agent per ospitare worker agent specifici del dominio

Riepilogo

Gli agenti del flusso di lavoro coordinano, adattano e allineano gli obiettivi in ambienti con più agenti. Ciò significa che gli agenti di intelligenza artificiale possono collaborare, adattarsi alle condizioni di runtime e fornire risultati complessi attraverso flussi di lavoro modulari e spiegabili.

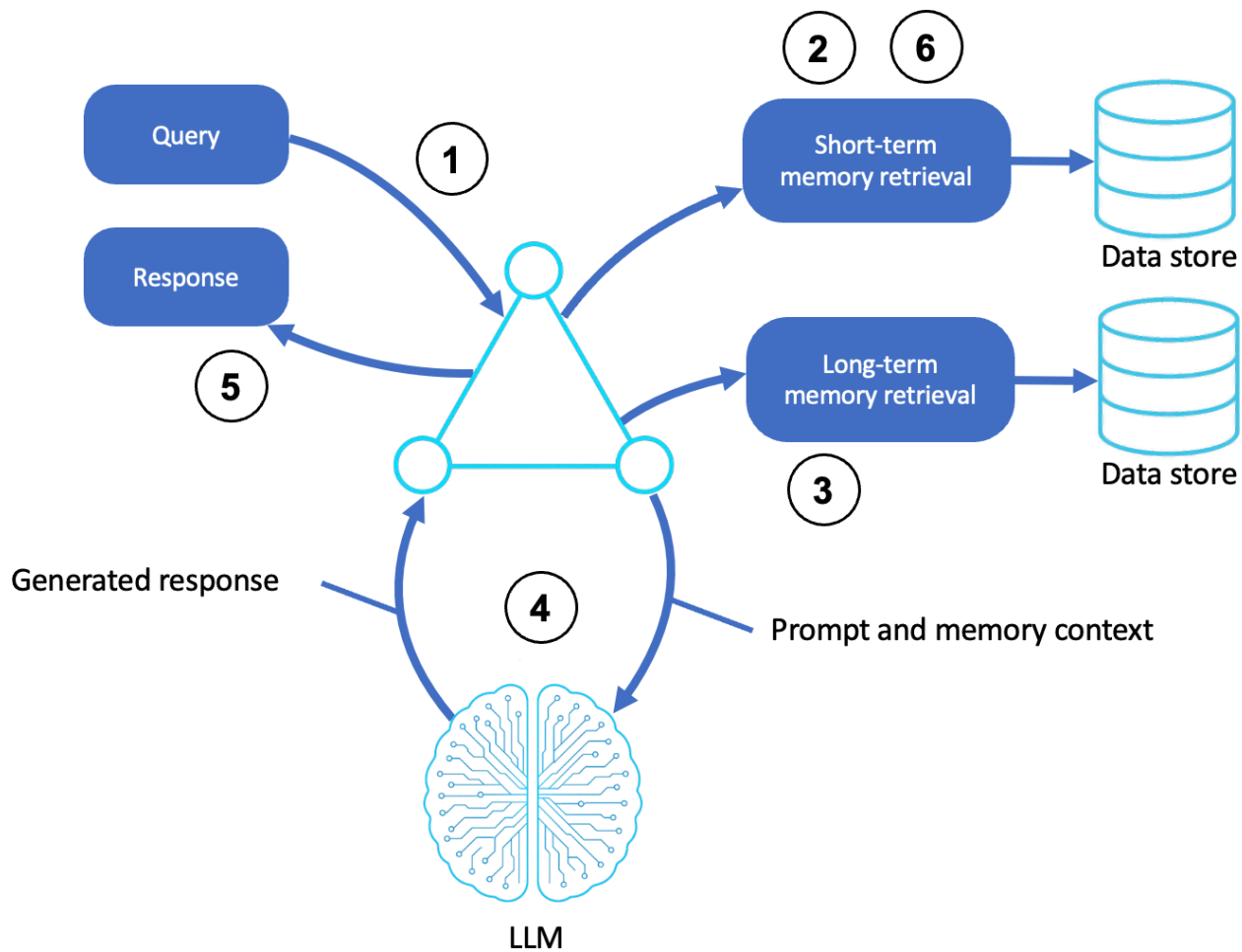
Memory-augmented agents

Memory-augmented gli agenti sono potenziati con la capacità di archiviare, recuperare e ragionare utilizzando la memoria a breve e lungo termine. Ciò consente loro di mantenere il contesto tra più attività, sessioni e interazioni, il che produce risposte più coerenti, personalizzate e strategiche.

A differenza degli agenti stateless, gli agenti con memoria aumentata si adattano facendo riferimento a dati storici, imparano dai risultati precedenti e prendono decisioni in linea con gli obiettivi, le preferenze e l'ambiente dell'utente.

Architecture

Un agente con memoria aumentata è illustrato nel diagramma seguente:



Description

1. Riceve input o eventi

- L'agente riceve una richiesta dell'utente o un evento di sistema. Può trattarsi di un testo, di un trigger dell'API o di una modifica ambientale.

2. Recupera la memoria a breve termine

- L'agente recupera la cronologia delle conversazioni recenti, il contesto delle attività o lo stato del sistema pertinenti alla sessione o al flusso di lavoro.

3. Recupera la memoria a lungo termine

- L'agente interroga la memoria a lungo termine (ad esempio, database vettoriali e archivi chiave-valore) per informazioni storiche, come le seguenti:
 - Preferenze dell'utente
 - Decisioni e risultati passati

- Concetti, riassunti o esperienze appresi

4. Motivi tramite il LLM

- Il contesto di memoria è incorporato nel prompt LLM e consente all'agente di ragionare in base sia agli input correnti che alle conoscenze precedenti.

5. Genera uscite

- L'agente produce una risposta, un piano o un'azione contestualmente consapevoli e personalizzati in base alla cronologia delle attività e agli input dell'utente.

6. Aggiorna la memoria

- Nuove informazioni, come obiettivi aggiornati, segnali di successo e fallimento e risposte strutturate, vengono archiviate per attività future.

Funzionalità

- Continuità della sessione tra conversazioni o eventi
- Persistenza degli obiettivi nel tempo
- Consapevolezza contestuale basata su uno stato in evoluzione
- Adattabilità basata su successi e fallimenti precedenti
- Personalizzazione in linea con le preferenze e la cronologia dell'utente

Casi di utilizzo comune

- Copiloti conversazionali che ricordano le preferenze dell'utente
- Agenti di codifica che tengono traccia delle modifiche alla codebase
- Agenti del flusso di lavoro che si adattano in base alla cronologia delle attività
- Gemelli digitali che si evolvono a partire dalla conoscenza del sistema
- Agenti di ricerca che evitano recuperi ridondanti

Implementazione di agenti con memoria aumentata

Utilizza i seguenti strumenti e Servizi AWS per gli agenti con memoria aumentata:

Livello di memoria

Servizio AWS

Scopo

Short-term	Contesto Amazon DynamoDB, Redis, Amazon Bedrock	Recupero rapido degli stati di interazione recenti
Long-term (strutturato)	Amazon Aurora, Amazon DynamoDB, Amazon Neptune	Fatti, relazioni e registri
Long-term (semantico)	OpenSearch, PostgreSQL, Pinecone	Embedding-based recupero (ovvero RAG)
Archiviazione	Simple Storage Service (Amazon S3)	Archiviazione di trascrizioni, memorie strutturate e file
Orchestrazione	AWS Lambda oppure AWS Step Functions	Gestione dell'iniezione di memoria e del ciclo di vita degli aggiornamenti
Ragionamento	Amazon Bedrock	Claude o Mistral antropico con richieste di memoria

Implementazione dei prompt iniettati dalla memoria

Per integrare la memoria nel ragionamento degli agenti, utilizzate una combinazione di stati strutturati e iniezione di contesto potenziata dal recupero:

- Includi lo stato più recente dell'agente e la cronologia recente dei dialoghi come input strutturato durante la creazione del prompt per il modello linguistico, in modo che possa ragionare con il contesto completo.
- Usa la generazione aumentata di recupero (RAG) per estrarre documenti o fatti pertinenti dalla memoria a lungo termine.
- Riassumi i piani, il contesto e le interazioni precedenti per la compressione e la pertinenza.
- Inserisci moduli di memoria esterni, come archivi vettoriali o log strutturati, durante l'inferenza per guidare il processo decisionale.

Riepilogo

Memory-augmented gli agenti mantengono la continuità del pensiero imparando dall'esperienza e ricordando il contesto dell'utente. Questi agenti superano l'intelligenza reattiva utilizzando la collaborazione a lungo termine, la personalizzazione e il ragionamento strategico. In termini di intelligenza artificiale agentica, la memoria consente agli agenti di comportarsi più come controparti digitali adattive e meno come strumenti stateless.

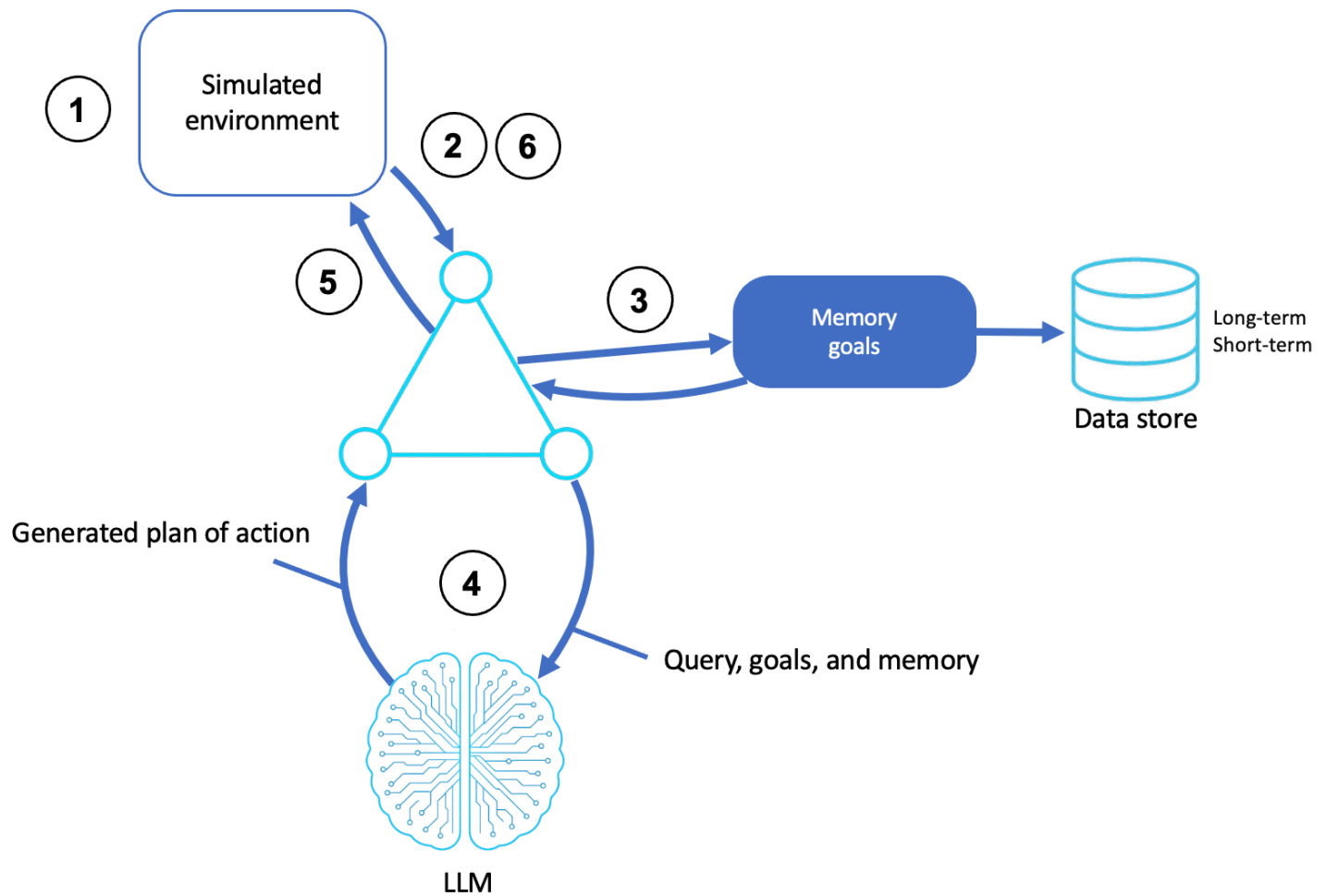
Agenti di simulazione e test

Gli agenti di simulazione e di test operano all'interno di ambienti virtualizzati o controllati in cui ragionano, agiscono e imparano. Questi agenti simulano il comportamento, modellano i risultati e addestrano le strategie in contesti ripetibili prima di applicarli agli ambienti del mondo reale.

Questo modello è utile per lo sviluppo iterativo, l'apprendimento per rinforzo (RL), la valutazione decisionale autonoma e i test comportamentali emergenti. Gli agenti di simulazione spesso operano a circuito chiuso, ricevono feedback dall'ambiente e adattano il loro comportamento di conseguenza, il che li rende fondamentali per attività che coinvolgono il ragionamento spaziale, il controllo in tempo reale o la dinamica di sistemi complessi.

Architecture

Il diagramma seguente mostra un agente di simulazione o di test:



Description

1. Avvia un ambiente

- L'agente avvia un ambiente simulato (ad esempio, un mondo 3D, un motore fisico, una sandbox CLI o un flusso di dati sintetici).
- L'agente viene caricato nell'ambiente con un'attività, un obiettivo o una politica iniziali.

2. Percepisce l'agente

- L'agente percepisce lo stato attuale tramite telemetria di simulazione (ad esempio, emulazione di sensori, fotocamera virtuale e registri strutturati).

3. Recupera obiettivo e memoria

- L'agente recupera l'obiettivo assegnato, le istruzioni sullo scenario o l'obiettivo contestuale.
- Può anche recuperare la memoria precedente, inclusa la seguente:
 - Long-term strategie o politiche

- mappe ambientali o vincoli noti
- Successi o fallimenti passati derivanti da simulazioni simili

4. Motivi e piani

- Un LLM interpreta lo stato simulato, gli obiettivi del compito e le conoscenze apprese.
- Genera un piano d'azione o un comando di controllo.

5. Esegue azioni simulate

- L'agente esegue il piano, modifica lo stato, naviga nello spazio o interagisce con entità virtuali.

6. Impara

- L'agente valuta i risultati delle azioni
- A seconda della configurazione dell'agente, può effettuare le seguenti operazioni:
 - Esegui RL
 - Registra i risultati per future ottimizzazioni
 - Adatta le strategie in tempo reale

Funzionalità

- Funziona all'interno di ambienti sintetici o virtuali
- Supporta l'apprendimento per tentativi ed errori, il perfezionamento delle politiche e la modellazione del sistema
- Low-risk test del comportamento, della gestione degli errori e dei casi limite
- Consente l'analisi del comportamento degli agenti emergenti in configurazioni con più agenti
- Supporta sia il controllo a circuito chiuso che l'esplorazione umana nel ciclo

Casi di utilizzo comune

- Apprendimento rinforzato per robotica, droni e giochi
- Addestramento con veicoli autonomi su strade virtuali
- Interfacce utente o CLI simulate per scenari da DevOps testare
- Esperimenti comportamentali emergenti nelle simulazioni sociali
- Convalida di sicurezza della logica decisionale prima della produzione

Guida all'implementazione

È possibile creare un agente di simulazione e test bed utilizzando i seguenti strumenti e Servizi AWS

Componente	Servizio AWS	Scopo
Ambiente	Amazon ECS, Amazon EC2 o un simulatore personalizzato in SageMaker Amazon Studio Lab	Esegui mondi virtuali (Gazebo, Unity, Unreal) o CLI sandbox
Logica dell'agente	Amazon Bedrock SageMaker, Amazon o AWS Lambda	LLM-based pianificatori o agenti RL
Circuito di feedback	Amazon SageMaker Reinforcement Learning CloudWatch, Amazon o log personalizzati	Monitoraggio dei premi, punteggio dei risultati e registrazione del comportamento
Memoria e replay	Amazon S3, Amazon DynamoDB o Amazon RDS	Dati persistenti sullo stato, sulla cronologia degli episodi o sullo scenario
Visualizzazione	CloudWatch Pannelli di controllo Amazon o notebook Amazon SageMaker	Osserva le modifiche alle politiche, i risultati e le metriche di formazione

Le seguenti sono applicazioni aggiuntive:

- [AWS SimSpace Weaver](#) per simulazioni spaziali su larga scala
- [AWS IoT Core](#) per testare dispositivi ombra
- [Amazon SageMaker Experiments](#) per la valutazione e il benchmarking degli agenti

Riepilogo

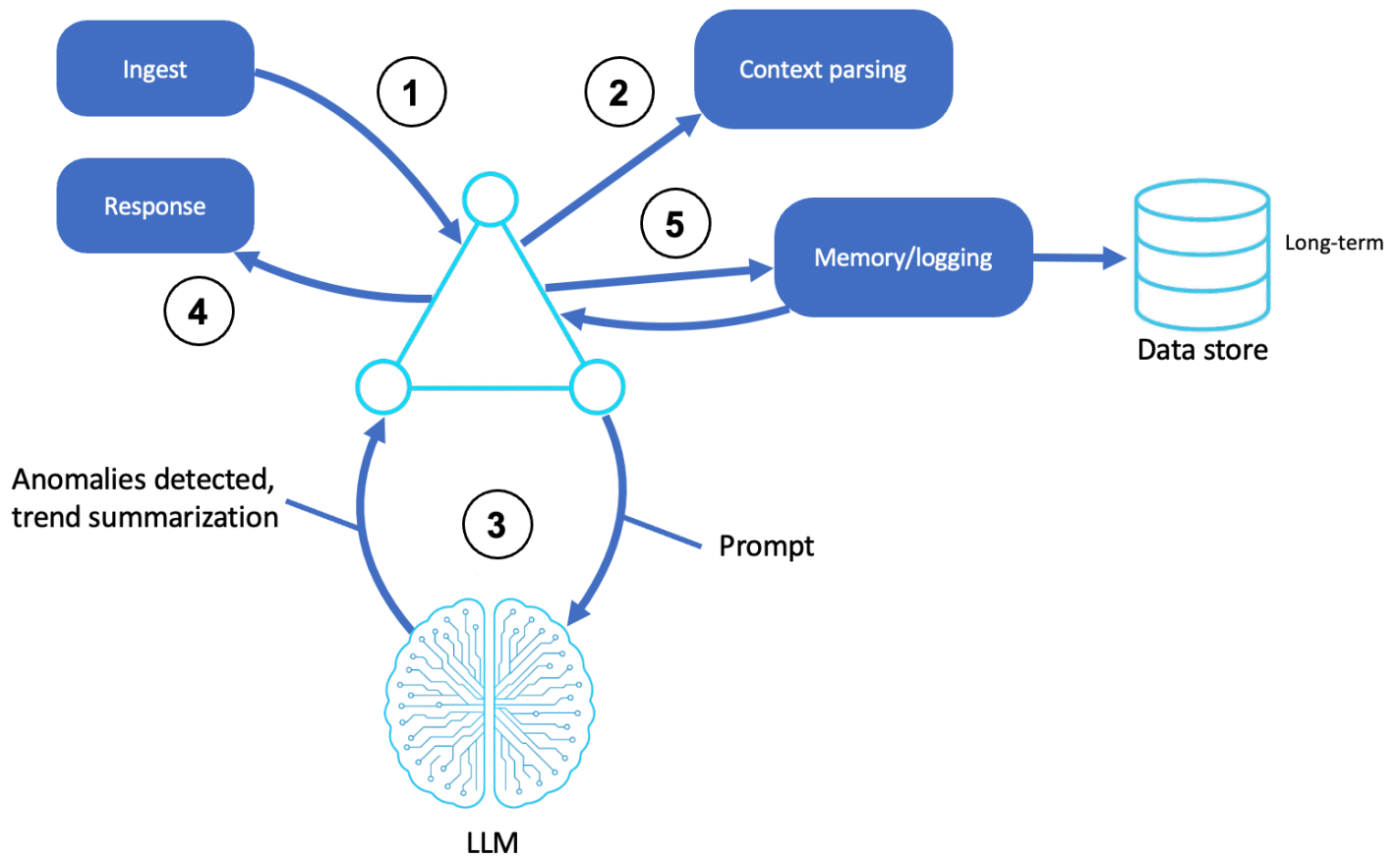
Gli agenti di simulazione e di test servono per l'esplorazione strutturata prima di essere implementati nei sistemi di produzione. Utilizzate questi agenti per addestrare politiche di navigazione autonome, testare i processi aziendali in ambienti sintetici e valutare gli sciami per i modelli di coordinamento.

Agenti osservatori e di monitoraggio

Gli osservatori e gli agenti di monitoraggio osservano passivamente i sistemi, gli ambienti e le interazioni per rilevare modelli, generare approfondimenti e innescare azioni. In qualità di osservatori intelligenti, migliorano gli avvisi, la diagnostica e gli audit senza avviare direttamente il comportamento.

Questi agenti eccellono laddove il monitoraggio tradizionale manca di adattabilità o ragionamento, in particolare per il AI-in-the-loop monitoraggio, il rilevamento delle anomalie, la supervisione della conformità e l'intelligence sulla sicurezza. Gli agenti osservatori sono ascoltatori di eventi che monitorano continuamente la telemetria del sistema e le interazioni con gli utenti. L'agente dipende dalla percezione, dall'interpretazione e dall'escalation o dalla segnalazione condizionale.

Architecture



Description

1. Acquisisci telemetria

- L'agente riceve input da una o più fonti di sistema, come le seguenti:
 - Registri (applicazione, infrastruttura, sicurezza)
 - Metriche (prestazioni, latenza, utilizzo)
 - Eventi (chiamate API, azioni dell'utente, dati dei sensori)

2. Analizza il contesto

- L'input non elaborato viene analizzato, strutturato e arricchito con metadati, come timestamp, identità dell'attore, stato del sistema e ID di traccia.

3. Motivi per utilizzare LLM

- L'agente utilizza un LLM o un modulo logico per interpretare gli input analizzati identificando le anomalie, riepilogando le tendenze e correlandoli tra tracce distribuite o finestre temporali.

4. Classificare o avvisare

- L'agente determina se il comportamento osservato giustifica quanto segue:
 - Un avviso o un'escalation
 - Un rapporto o un aggiornamento del pannello di controllo
 - Un trigger di risposta (ad esempio, riparazione automatica e applicazione delle politiche)

5. Registra la memoria o i loop di feedback

- L'agente archivia eventi e decisioni per l'apprendimento a lungo termine, gli audit o riferimenti futuri per altri agenti.

Funzionalità

- Passivo e non invasivo (l'agente non agisce direttamente)
- Altamente scalabile e asincrono
- AI-driven correlazione tra segnali rumorosi o distribuiti
- Supporta audit, conformità e informazioni in tempo reale
- Può alimentare agenti a valle o flussi di lavoro umani

Casi di utilizzo comune

- AI-augmented osservabilità per microservizi e API
- Monitoraggio della deriva del modello, della violazione delle policy o del comportamento fuori banda
- Analisi delle attività o riepiloghi delle interazioni con i clienti
- Agenti di revisione del codice che monitorano gli impegni o le implementazioni
- Monitoraggio dei registri di sicurezza o conformità mediante il ragionamento LLM

Guida all'implementazione

È possibile creare un osservatore e un agente di monitoraggio utilizzando i seguenti strumenti e Servizi AWS

Componente

Servizio AWS

Scopo

Ingestione di eventi	Amazon EventBridge, Amazon CloudWatch Logs, Amazon Kinesis, Amazon S3	Inserisci telemetria strutturata e non strutturata
Pre-elaborazione	AWS Lambda, AWS Glue, AWS Step Functions	Trasforma i dati grezzi in prompt strutturati
Motore di ragionamento	Amazon Bedrock, Amazon SageMaker, AWS Lambda	Analizza gli eventi, classifica i comportamenti, genera approfondimenti
Archiviazione e memoria	Amazon S3, Amazon DynamoDB, OpenSearch	Osservazioni, riepiloghi e risultati persistenti
Avvisi e intensificazione	Amazon SNS, AWS AppFabric Amazon EventBridge	Attiva sistemi o agenti downstream

Le seguenti sono applicazioni aggiuntive:

- [AWS Security Hub CSPM](#) per il monitoraggio dei registri di sicurezza
- [Amazon Quick](#) per la visualizzazione degli output degli agenti

Riepilogo

Gli osservatori e gli agenti di monitoraggio tengono traccia dei sistemi e dei comportamenti in tempo reale. Rilevano anomalie, controllano la sicurezza e raccolgono informazioni operative identificando modelli che gli umani o le regole potrebbero trascurare. Questa funzionalità consente di creare sistemi in grado di adattarsi alle mutevoli condizioni e di prendere decisioni basate su un'analisi completa dei dati.

Multi-agent collaborazione

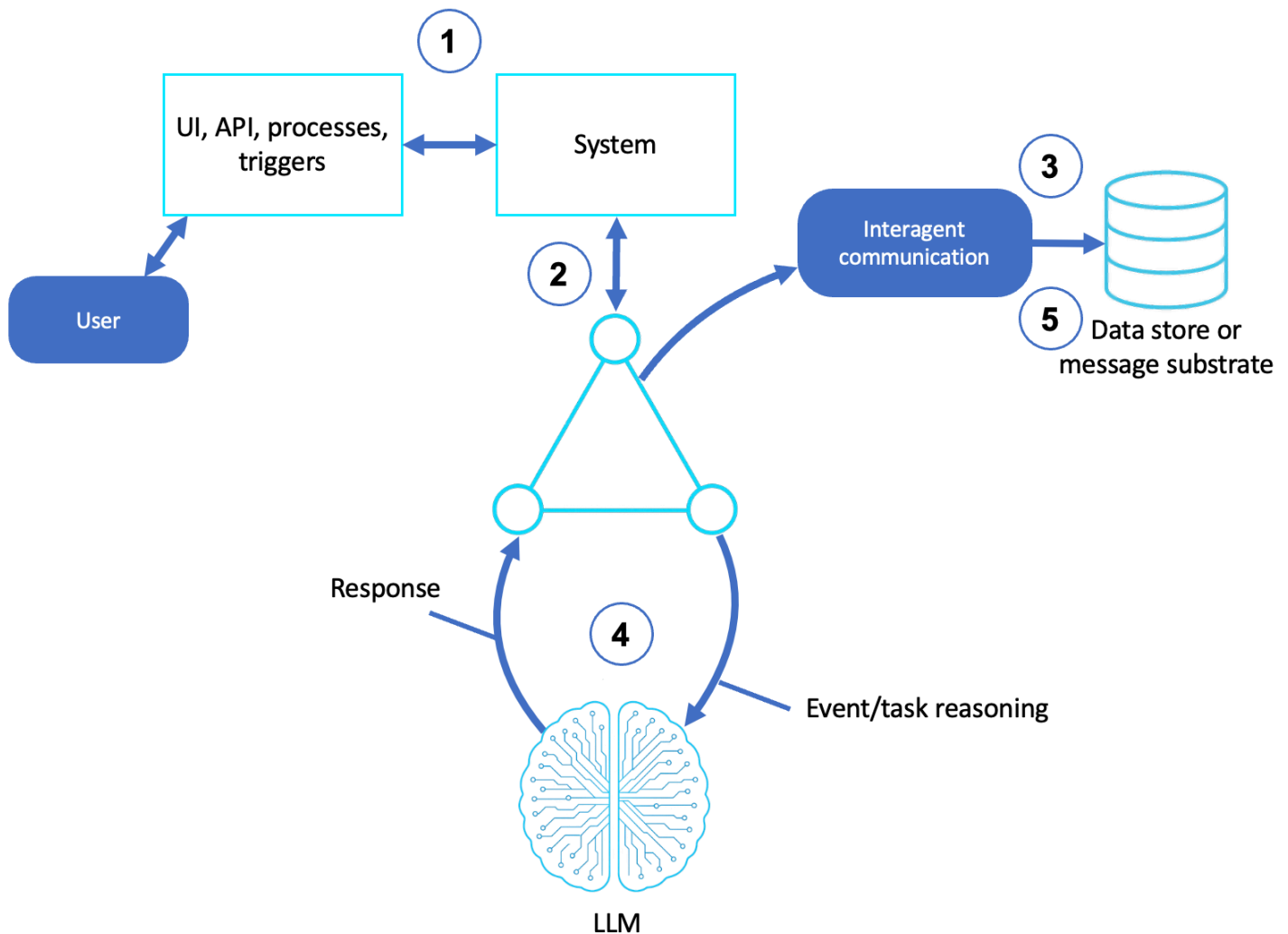
Multi-agent la collaborazione si riferisce a un modello in cui più agenti autonomi, ciascuno con un ruolo, una specializzazione o un obiettivo distinti, negoziano per risolvere compiti complessi. Questi agenti possono operare indipendentemente o con altri agenti condividendo informazioni, dividendo le responsabilità e ragionando collettivamente verso un obiettivo.

Questo modello è diverso dagli agenti del flusso di lavoro, che coordinano e delegano centralmente le attività agli agenti subordinati in un flusso strutturato. Al contrario, la collaborazione tra più agenti enfatizza la coordinazione peer-to-peer o emergente, abilitando l'adattabilità, il parallelismo e la divisione della cognizione. La tabella seguente mette a confronto la collaborazione tra più agenti e gli agenti del flusso di lavoro:

Funzionalità	Agenti di workflow	Scopo
Controllo	Coordinatore centralizzato	Peer decentralizzati, distribuiti o basati sui ruoli
Interazione	Un agente delega e monitora l'esecuzione	Più agenti negoziano, condividono e si adattano
Progettazione	Sequenza predefinita di attività	Distribuzione delle attività emergente e flessibile
Coordinamento	Orchestrazione procedurale	Interazioni cooperative o competitive
Casi d'uso	Automazione dei processi aziendali	Ragionamento complesso , esplorazione e strategie emergenti

Architecture

Il diagramma seguente mostra la collaborazione tra più agenti:



Description

1. Avvia un'attività

- Un utente o un sistema emette un obiettivo o un problema di alto livello.
- Un agente «manager» o un contesto iniziale definisce l'obiettivo.

2. Assegna o scopre i ruoli

- Gli agenti si autoassegnano (logica o ragionamento simbolico) o sono delegati (mediatori di eventi) ad altri ruoli, come pianificatore, ricercatore, esecutore, critico o spiegatore.

3. Comunica con altri agenti

- Gli agenti comunicano tramite memoria condivisa, code di messaggistica o concatenamento di prompt.

- Possono discutere, interrogarsi o proporsi attività secondarie l'un l'altro.

4. Utilizza un ragionamento specializzato

- Ogni agente utilizza il proprio modello o la propria logica di dominio per risolvere una parte del problema.
- Gli agenti possono utilizzare LLM con prompt e memoria specifici per ruolo.

5. Coordina i risultati o gli obiettivi

- Gli agenti sintetizzano i contributi in una risposta, un piano o un'azione finale.
- (Facoltativo) Un agente supervisore può convalidare o riassumere l'output sintetizzato.

Funzionalità

- Peer-level agenti con ruoli o competenze specializzati
- Comportamento emergente attraverso la comunicazione o la negoziazione
- Elaborazione parallela di problemi complessi o sfaccettati
- Supporta la deliberazione, l'autocorrezione e l'iterazione riflessiva
- Modella le dinamiche sociali, la collaborazione scientifica o i ruoli dei team aziendali

Casi di utilizzo comune

- Team di ricerca autonomi (agente di ricerca, riepilogo e validatore)
- Sviluppo di software (pianificatore, programmatore e tester)
- Modellazione di scenari aziendali (finanze, politiche e conformità)
- Negoziazione, offerta o ragionamento multipartitico
- Attività multimodali (immagine, testo e logica)

Guida all'implementazione

È possibile creare un sistema multiagente utilizzando i seguenti strumenti e: Servizi AWS

Componente

Servizio AWS

Scopo

Hosting per agenti	Amazon Bedrock, Amazon SageMaker, AWS Lambda	Ospita singoli agenti LLM-driven
Livello di comunicazione	Amazon SQS, Amazon EventBridge AWS AppFabric	Messaggistica e coordinamento tra agenti
Memoria condivisa	Amazon DynamoDB, Amazon S3 o OpenSearch	Multi-agent sistema di memoria o lavagna
Livello di orchestrazione	AWS Step Functions, condutture AWS Lambda	Logica di kickoff, timeout, fallback e ripetizione
Identificazione dell'agente	Agenti Amazon Bedrock (definiti dal ruolo) e API converse di AWS AppConfig Amazon Bedrock (agenti esterni ad Amazon Bedrock)	Role-based invocazione di strumenti o agenti e applicazione dei limiti
Interazione emergente	Amazon EventBridge Pipeline o registri degli agenti	Abilita il routing o l'escalation dinamica delle attività

Riepilogo

Multi-agent la collaborazione distribuisce le attività di risoluzione dei problemi tra agenti modulari e basati sui ruoli. A differenza dell'orchestrazione del flusso di lavoro, i modelli di collaborazione utilizzano intelligenza, resilienza e scalabilità emergenti che rispecchiano il modo in cui gli esseri umani risolvono i problemi. È particolarmente utile per domini aperti, attività creative, ragionamento multimodale e ambienti che traggono vantaggio da prospettive diverse.

Conclusioni

I modelli discussi in precedenza illustrano gli approcci fondamentali alle implementazioni nel mondo reale dell'intelligenza artificiale agentica. Dal ragionamento di base all'intelligenza potenziata dalla memoria, ogni modello è configurato in modo univoco per la percezione, la cognizione e l'azione, basate su autonomia, asincronia e azione.

Questi modelli condividono vocabolari e modelli tecnici per la creazione di sistemi intelligenti e orientati agli obiettivi. Sia che un pattern sia incorporato in un'interfaccia utente, orchestrato tramite servizi cloud o coordinato tra team di agenti, ogni pattern è adattabile e modulare.

Da asporto

- Gli schemi degli agenti sono componibili: la maggior parte degli agenti del mondo reale combina due o più schemi (ad esempio, un agente vocale con ragionamento e memoria basati su strumenti).
- La progettazione degli agenti è contestuale: scegli i modelli in base alla superficie di interazione, alla complessità delle attività, alla tolleranza alla latenza e ai vincoli specifici del dominio.
- AWS l'implementazione nativa è realizzabile: con Amazon Bedrock SageMaker, Amazon e le architetture event-driven AWS Lambda AWS Step Functions, ogni modello di agente può essere fornito su larga scala.

Flussi di lavoro LLM

Nei modelli degli agenti, abbiamo esplorato i modelli comuni degli agenti di intelligenza artificiale, ciascuno basato su una serie di funzionalità modulari: percezione, azione, apprendimento e cognizione. Al centro del modulo cognitivo di molti modelli di agenti c'è un ampio modello linguistico (LLM) in grado di ragionare, pianificare e prendere decisioni. Tuttavia, invocare un LLM da solo non è sufficiente per produrre un comportamento intelligente e mirato.

Per eseguire attività complesse in modo affidabile, gli agenti devono incorporare il LLM in un flusso di lavoro strutturato, in cui le funzionalità del modello sono potenziate da strumenti, memoria, cicli di pianificazione e logica di coordinamento. Questi flussi di lavoro LLM consentono a un agente di suddividere gli obiettivi, indirizzare le sottoattività, chiamare servizi esterni, riflettere sui risultati e coordinarsi con altri agenti.

Questo capitolo introduce i modelli di progettazione di base per la creazione di moduli LLM-driven cognitivi robusti, estensibili e intelligenti, organizzati attorno a flussi di lavoro riutilizzabili.

In questa sezione

- [LLM-augmented Panoramica della cognizione](#)
- [Flusso di lavoro per il concatenamento rapido](#)
- [Flusso di lavoro per il routing](#)
- [Flusso di lavoro per la parallelizzazione](#)
- [Flusso di lavoro per l'orchestrazione](#)
- [Flusso di lavoro per valutatori e cicli Reflect-Refine](#)
- [Conclusioni](#)

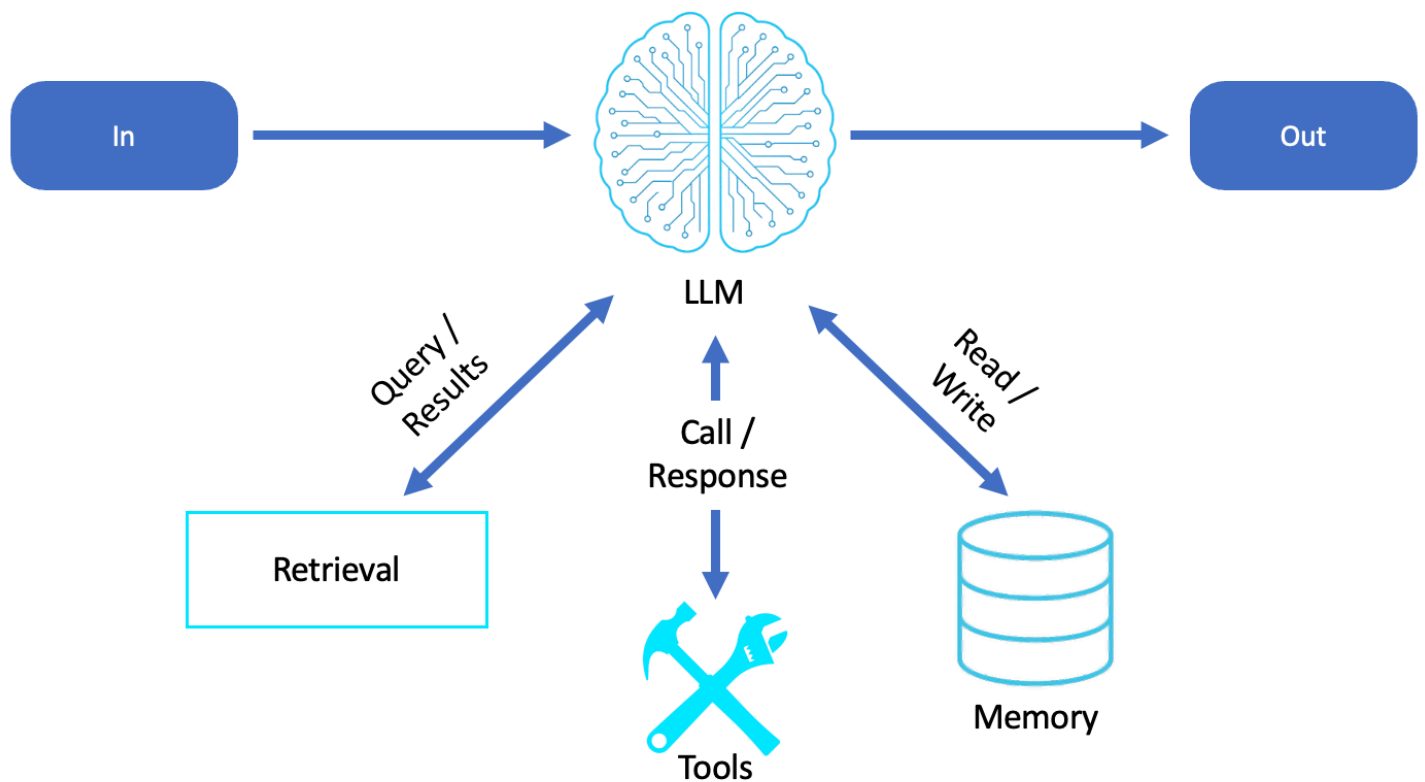
LLM-augmented Panoramica della cognizione

Fondamentalmente, il modulo cognitivo di un agente software può essere visto come un LLM avvolto da potenziamenti. L'agente può utilizzare i seguenti elementi costitutivi per ragionare in modo efficace all'interno del suo ambiente:

- Richiesta: inquadra l'input utilizzando contesto, istruzioni, esempi e memoria

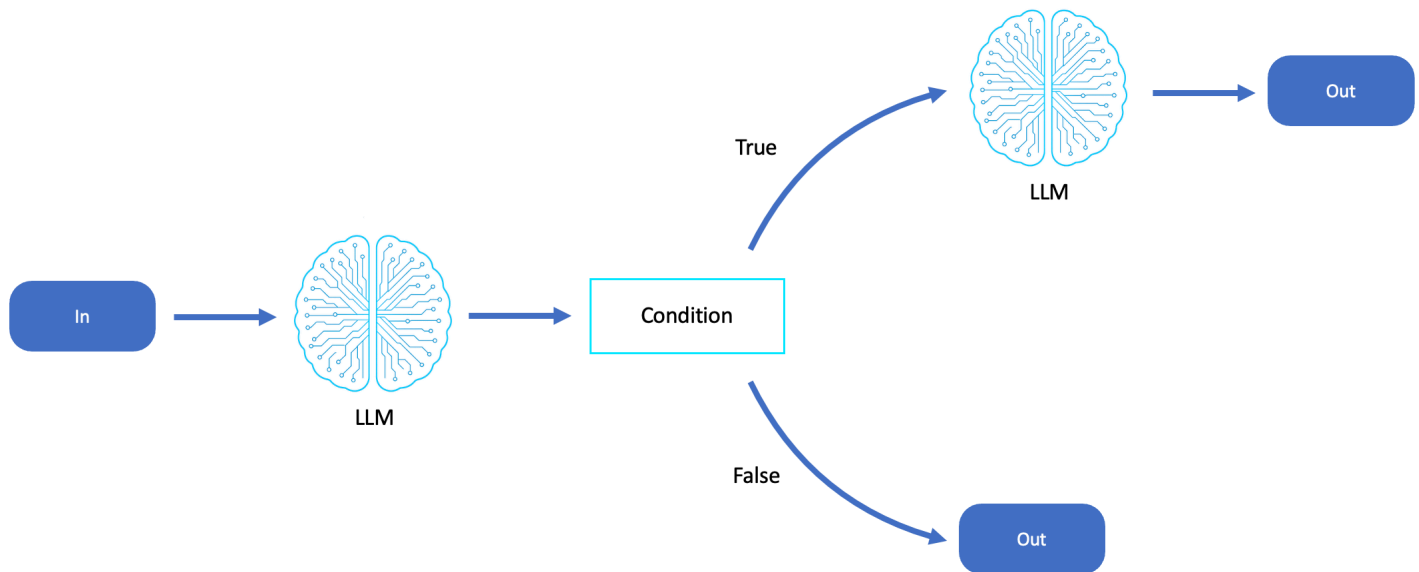
- **Recupero:** fornitura di conoscenze aggiornate o specifiche del dominio al prompt LLM tramite la ricerca vettoriale o la memoria semantica, ad esempio tramite la generazione aumentata di recupero (RAG)
- **Utilizzo degli strumenti:** consentire all'LLM di richiamare API o richiamare funzioni per recuperare o agire in base alle informazioni
- **Memoria:** incorporazione di uno stato persistente o basato sulla sessione nel ciclo di ragionamento, utilizzando database strutturati o riepiloghi contestuali

Questi miglioramenti sono composti da flussi di lavoro che definiscono il modo in cui l'LLM viene utilizzato nel tempo e tra le attività, trasformandolo da un motore stateless a un agente di ragionamento dinamico.



Flusso di lavoro per il concatenamento rapido

Il prompt chaining scompone le attività complesse in una sequenza di passaggi, in cui ogni passaggio è una chiamata LLM discreta che elabora o si basa sull'output di quella precedente.



Il flusso di lavoro di concatenamento dei prompt è adatto per scenari in cui le attività possono essere suddivise logicamente in fasi di ragionamento sequenziali e in cui gli output intermedi determinano la fase successiva. Eccelle nei flussi di lavoro che richiedono un pensiero strutturato, una trasformazione progressiva o un'analisi su più livelli, come la revisione dei documenti, la generazione di codice, l'estrazione della conoscenza e il perfezionamento dei contenuti.

Description

- La complessità dell'attività supera la finestra contestuale o la profondità di ragionamento di una singola chiamata LLM.
- I risultati di una fase (ad esempio, analisi, riepilogo o pianificazione) diventano input per una successiva fase decisionale o di generazione.
- Sono necessari trasparenza e controllo in tutte le fasi di ragionamento (ad esempio, risultati intermedi verificabili).
- Vuoi inserire una logica di convalida, filtraggio o arricchimento esterna tra i passaggi.
- È ideale per gli agenti che operano in cicli di ragionamento in stile pipeline, come agenti di ricerca, assistenti editoriali, sistemi di pianificazione e copiloti multistadio.

Funzionalità

- Catene lineari o ramificate di chiamate LLM
- Risultati intermedi passati come input strutturato o incorporati nei prompt di follow-up

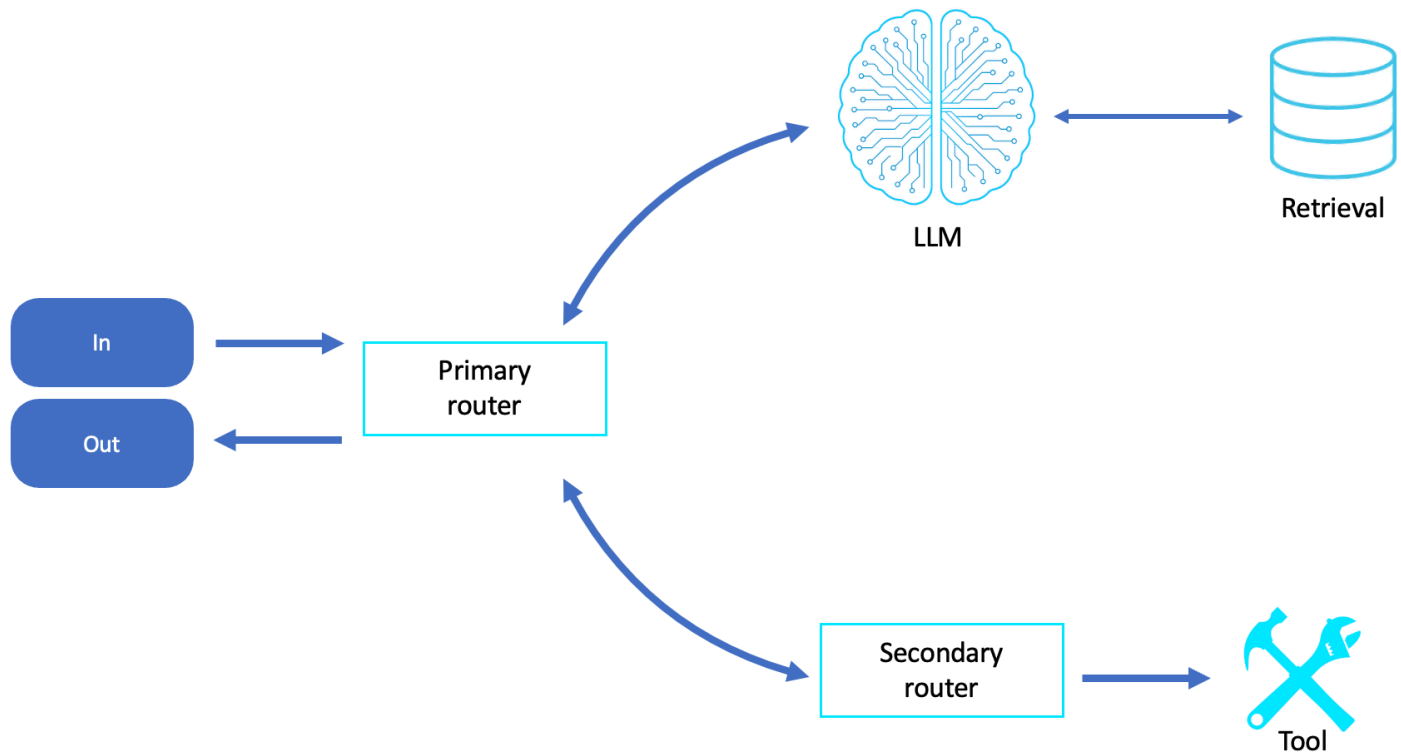
- Può essere orchestrato con o con corridori specifici per un agente AWS Step Functions AWS Lambda

Casi di utilizzo comune

- Attività di ragionamento in più fasi (ad esempio, «riepilogo, riscrittura critica»)
- Assistenti di ricerca che sintetizzano risultati stratificati (ad esempio, «cerca, estrai fatti, rispondi a una domanda»)
- Pipeline di generazione del codice («generate plan write code test code explain output»)

Flusso di lavoro per il routing

Nel modello di routing, un classificatore o un agente router utilizza un LLM per interpretare l'intento o la categoria di una query, quindi indirizza l'input a un'attività o un agente downstream specializzato.



Il flusso di lavoro di routing viene utilizzato in scenari in cui un agente deve classificare rapidamente l'intento di input, il tipo di attività o il dominio e quindi delegare la richiesta a un subagente, uno strumento o un flusso di lavoro specializzato. È particolarmente utile negli agenti di funzionalità,

come quelli che fungono da assistenti generali, nelle porte di accesso alle funzioni aziendali o nelle interfacce AI rivolte agli utenti che si estendono su più domini.

Il routing è particolarmente efficace quando:

- Classificazione delle richieste per una serie di attività (ad esempio ricerca, riepilogo, prenotazione, calcoli).
- Gli input devono essere preelaborati o normalizzati prima di accedere a flussi di lavoro più specializzati.
- Tipi di input diversi (ad esempio, immagini anziché testo, query strutturate e non strutturate) richiedono una gestione personalizzata.
- Un agente funge da centralino conversazionale, delegando attività ad agenti specializzati o microservizi.
- Questo flusso di lavoro è comune nei copiloti specifici del dominio, nei bot di assistenza clienti, nei router di servizi aziendali e negli agenti multimodali, dove il dispacciamento intelligente determina sia la qualità che l'efficienza del comportamento degli agenti.

Funzionalità

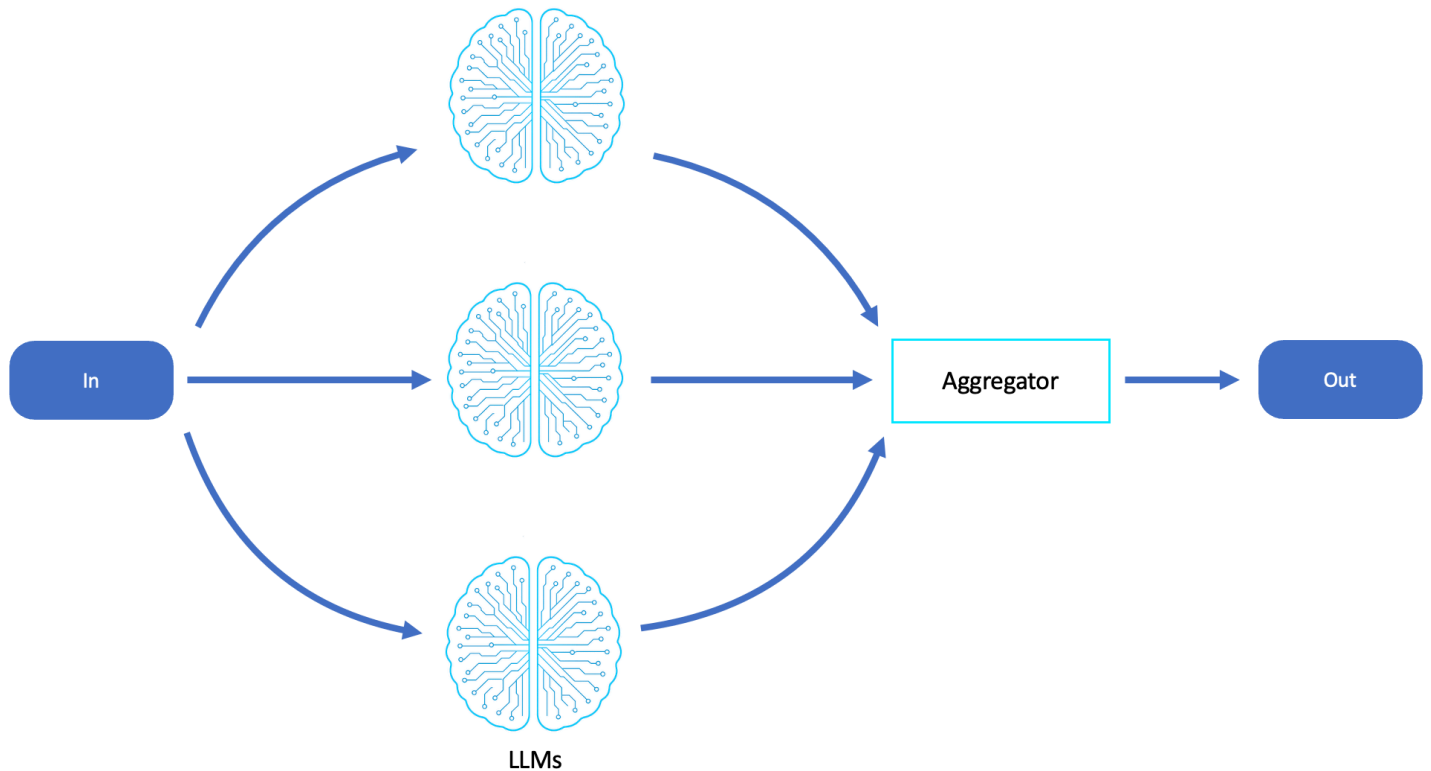
- Un LLM di primo passaggio funge da dispatcher
- I percorsi possono richiamare flussi di lavoro distinti o persino altri modelli di agenti
- Supporta l'espansione modulare delle funzionalità

Casi di utilizzo comune

- Assistenti multidominio («è una questione legale, medica o finanziaria?»)
- Alberi decisionali migliorati con il ragionamento LLM
- Selezione dinamica degli strumenti (ad esempio, ricerca e generazione di codice)

Flusso di lavoro per la parallelizzazione

Questo flusso di lavoro prevede la suddivisione di un'attività in sottoattività indipendenti che possono essere gestite contemporaneamente da più chiamate o agenti LLM. Gli output vengono quindi aggregati a livello di codice e sintetizzati in un risultato.



Il flusso di lavoro di parallelizzazione viene utilizzato quando un'attività può essere suddivisa in sottoattività indipendenti e non sequenziali che possono essere elaborate contemporaneamente, migliorando significativamente l'efficienza, la produttività e la scalabilità. È particolarmente efficace negli spazi problematici ad alto contenuto di dati, orientati ai batch o multiprospettici in cui l'agente deve analizzare o generare contenuti su più input.

La parallelizzazione è particolarmente efficace quando:

- Le sottoattività non dipendono dai reciproci risultati intermedi, pertanto possono essere eseguite in parallelo senza coordinamento.
- Un'attività prevede la ripetizione dello stesso processo di ragionamento su più elementi (ad esempio, riepilogando più documenti o valutando un elenco di opzioni).
- Molteplici ipotesi o prospettive vengono esplorate in parallelo per promuovere la diversità, la creatività o la solidità.
- È necessario ridurre la latenza per le richieste ad alto volume o ad alta frequenza tramite l'esecuzione simultanea di LLM.
- Questo flusso di lavoro viene comunemente utilizzato negli agenti di elaborazione di documenti, nei motori di sondaggi o di confronto, nei riepiloghi di batch, nei brainstorming multiagente e nelle

attività di classificazione o etichettatura scalabili, specialmente laddove il ragionamento rapido e parallelo rappresenta un vantaggio in termini di prestazioni.

Funzionalità

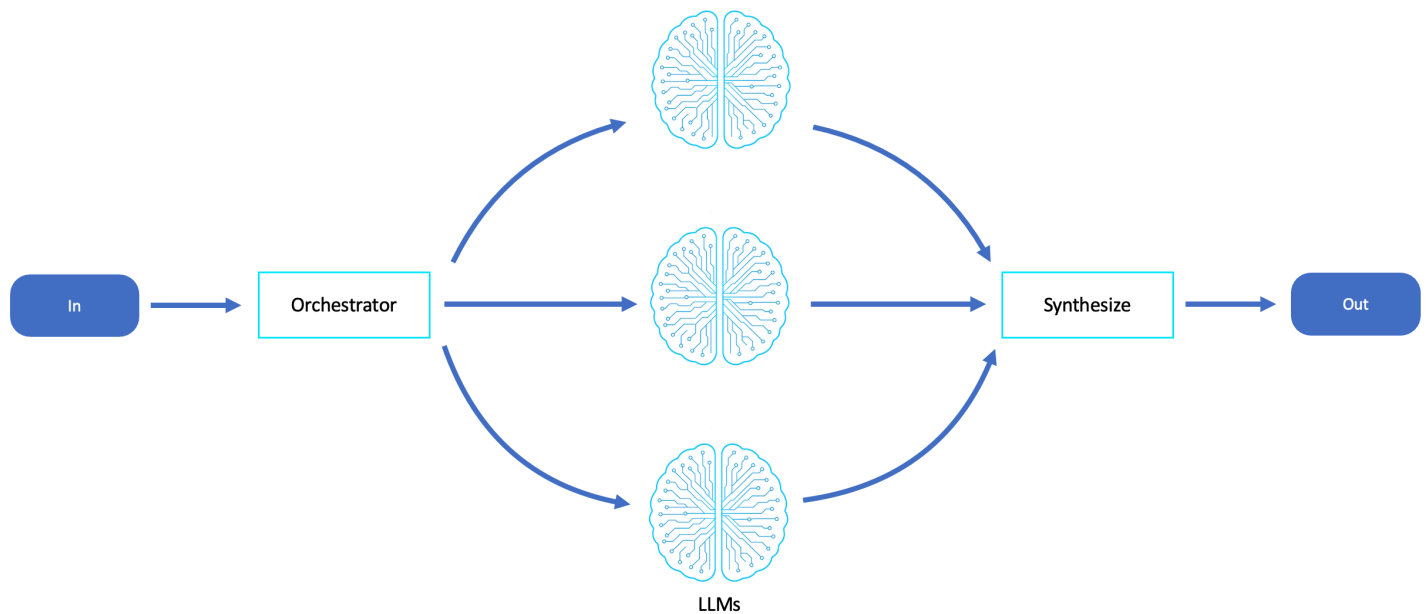
- Esecuzione parallela di attività LLM (utilizzando o uno stato della mappa) AWS Lambda AWS Fargate AWS Step Functions
- Richiede l'allineamento, la convalida o la deduplicazione dei risultati in fase di sintesi
- Well-suited per loop di agenti stateless

Casi di utilizzo comune

- Analisi parallela di più documenti o prospettive
- Generazione di bozze, riepiloghi o piani diversi
- Accelerazione della produttività tra i lavori in batch

Flusso di lavoro per l'orchestrazione

Un agente di orchestrazione centrale utilizza un LLM per pianificare, scomporre e delegare le sottoattività ad agenti o modelli di lavoro specializzati, ciascuno con un ruolo o un'esperienza di dominio specifici. Ciò rispecchia le strutture dei team umani e supporta il comportamento emergente tra più agenti.



Il flusso di lavoro di orchestrazione è ideale per scenari complessi, gerarchici o multidisciplinari, che richiedono una scomposizione strutturata e un'esecuzione specializzata. È particolarmente adatto per attività che richiedono la divisione del lavoro, in cui i diversi sottocomponenti di un'attività vengono gestiti al meglio da agenti con capacità, conoscenze o set di strumenti distinti.

Questo flusso di lavoro è particolarmente efficace quando:

- Le attività possono essere suddivise in sottoattività che variano per ambito, tipo o ragionamento (ad esempio, pianificazione, ricerca, implementazione e test).
- Un LLM o un meta-agente deve coordinare altri agenti, monitorare i progressi e sintetizzare i risultati.
- Desiderate modularizzare le responsabilità degli agenti, abilitando la scalabilità, il riutilizzo e l'ottimizzazione specializzata.
- Il sistema richiede un comportamento basato sui ruoli, che imita il modo in cui i team umani (ad esempio, project manager, sviluppatori e revisori) operano in collaborazione.

L'orchestrazione è ideale per agenti di pianificazione a più turni, copiloti di sviluppo software, agenti di processo aziendali ed esecutori di progetto autonomi. È particolarmente utile quando si implementano sistemi multiagente che richiedono una suddivisione centralizzata delle attività ma una logica di esecuzione distribuita, che consente l'estensibilità e un comportamento più spiegabile tra i livelli degli agenti.

Funzionalità

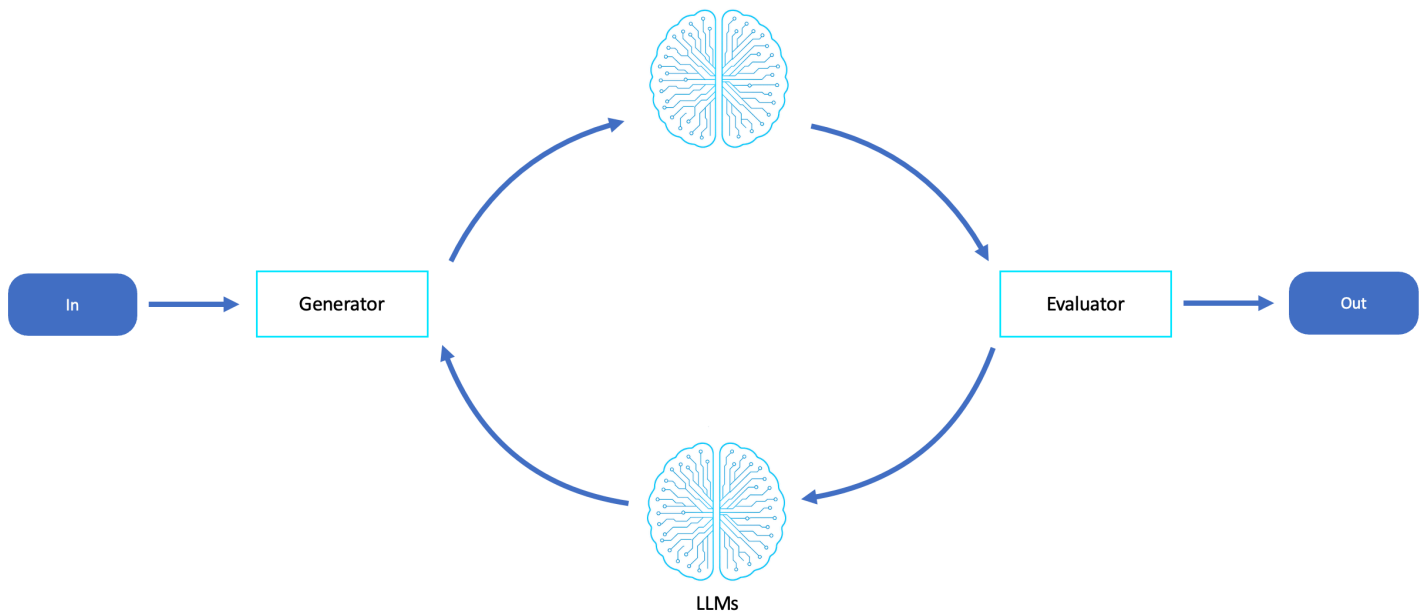
- Orchestrator esegue il meta-ragionamento degli obiettivi
- Gli agenti di lavoro possono includere l'accesso agli strumenti, la memoria o richieste specifiche del dominio
- Può essere gerarchico (ovvero delega di attività a più livelli)

Casi di utilizzo comune

- Responsabili di progetto, ricercatori coordinatori, scrittori e agenti di garanzia della qualità
- Copiloti di codifica che combinano pianificazione, esecuzione e test
- Agenti che supervisionano le toolchain o i modelli di accesso alle API

Flusso di lavoro per valutatori e cicli Reflect-Refine

Questo flusso di lavoro fornisce un ciclo di feedback in cui un LLM genera un risultato e un altro valuta o critica il risultato. Ciò promuove l'autoriflessione, l'ottimizzazione e i miglioramenti iterativi.



Il flusso di lavoro del valutatore è ideale per scenari in cui la qualità, la precisione e l'allineamento dell'output sono importanti e in cui la generazione a passaggio singolo è inaffidabile o insufficiente. Questo flusso di lavoro eccelle quando gli agenti devono autocriticare, iterare e perfezionare i propri

risultati, per soddisfare uno standard di correttezza più elevato o per esplorare alternative migliori basate sul feedback.

Questo flusso di lavoro è particolarmente efficace quando:

- L'output include metriche di qualità soggettive (ad esempio stile, tono e leggibilità) o criteri oggettivi (ad esempio, correttezza, sicurezza e prestazioni).
- L'agente deve ragionare sulla base di compromessi, valutare i vincoli o ottimizzare per raggiungere un obiettivo.
- Sono necessarie ridondanza e garanzia di qualità integrate, specialmente in domini regolamentati, rivolti ai clienti o creativi.
- Human-in-the-loop la revisione è costosa o non disponibile e si desidera una convalida autonoma.

Questo flusso di lavoro viene utilizzato per la generazione di contenuti, la sintesi e la revisione del codice, l'applicazione delle policy, il controllo dell'allineamento, l'ottimizzazione delle istruzioni e la postelaborazione RAG. È utile anche per gli agenti che si migliorano da soli, dove il feedback continuo aiuta a modellare risposte migliori nel tempo per creare cicli decisionali affidabili e autonomi.

Casi di utilizzo comune

- Red-team agenti rispetto agli agenti della squadra blu
- Agenti che generano, valutano e revisionano codice o piani
- Garanzia della qualità, rilevamento delle allucinazioni e applicazione dello stile

Funzionalità

- Supporta la generazione e la valutazione disaccoppiate utilizzando diversi modelli (ad esempio, Claude per la generazione e Mistral per la valutazione)
- Il feedback è strutturato e utilizzato per richiedere risultati rivisti
- Supporta più iterazioni o soglie di convergenza

Conclusioni

Gli LLM forniscono il nucleo cognitivo dei moderni agenti software, ma l'invocazione di modelli grezzi non è sufficiente per ottenere un'intelligenza mirata, solida e controllabile. Per passare dalla

generazione di output al ragionamento strutturato e al comportamento allineato agli obiettivi, i LLM devono essere incorporati in modelli di flusso di lavoro intenzionali che definiscono il modo in cui i modelli elaborano gli input, gestiscono i contesti e coordinano le azioni.

I flussi di lavoro LLM introducono le basi per creare il modulo cognitivo di un agente:

- Il concatenamento rapido suddivide il ragionamento complesso in fasi modulari e verificabili.
- Il routing consente una classificazione intelligente delle attività e una delega mirata.
- La parallelizzazione accelera la produttività e promuove ragionamenti diversificati.
- L'orchestrazione degli agenti struttura la collaborazione tra più agenti attraverso la scomposizione delle attività e l'esecuzione basata sui ruoli.
- Evaluator (reflect-refine loop) consente il miglioramento automatico, il controllo della qualità e il controllo dell'allineamento.

Ogni flusso di lavoro rappresenta uno schema componibile che può essere adattato alle esigenze dell'operatore, alla complessità dell'attività e alle aspettative dell'utente. Questi flussi di lavoro non si escludono a vicenda. Sono elementi costitutivi che vengono spesso combinati in architetture ibride che supportano il ragionamento dinamico, il coordinamento multiagente e l'affidabilità di livello aziendale.

Passando al prossimo capitolo sui modelli di flusso di lavoro agentici, questi flussi di lavoro LLM riappariranno come strutture integrate all'interno di sistemi più grandi, che supportano la delega degli obiettivi, l'orchestrazione degli strumenti, i cicli decisionali e l'autonomia del ciclo di vita. La padronanza di questi flussi di lavoro LLM è essenziale per progettare agenti software che non si limitino a prevedere il testo, ma che ragionino, si adattino e agiscano in modo mirato.

Modelli di flusso di lavoro agentici

I modelli di flusso di lavoro agentici integrano agenti software modulari con flussi di lavoro LLM (Structured Large Language Model), consentendo ragionamenti e azioni autonomi. Sebbene ispirati alle tradizionali architetture serverless e basate sugli eventi, questi modelli spostano la logica di base dal codice statico agli agenti, garantendo una maggiore adattabilità e un processo decisionale contestuale. LLM-augmented Questa evoluzione trasforma le architetture cloud convenzionali da sistemi deterministici a sistemi capaci di interpretazione dinamica e aumento intelligente, pur mantenendo i principi fondamentali di scalabilità e reattività.

In questa sezione

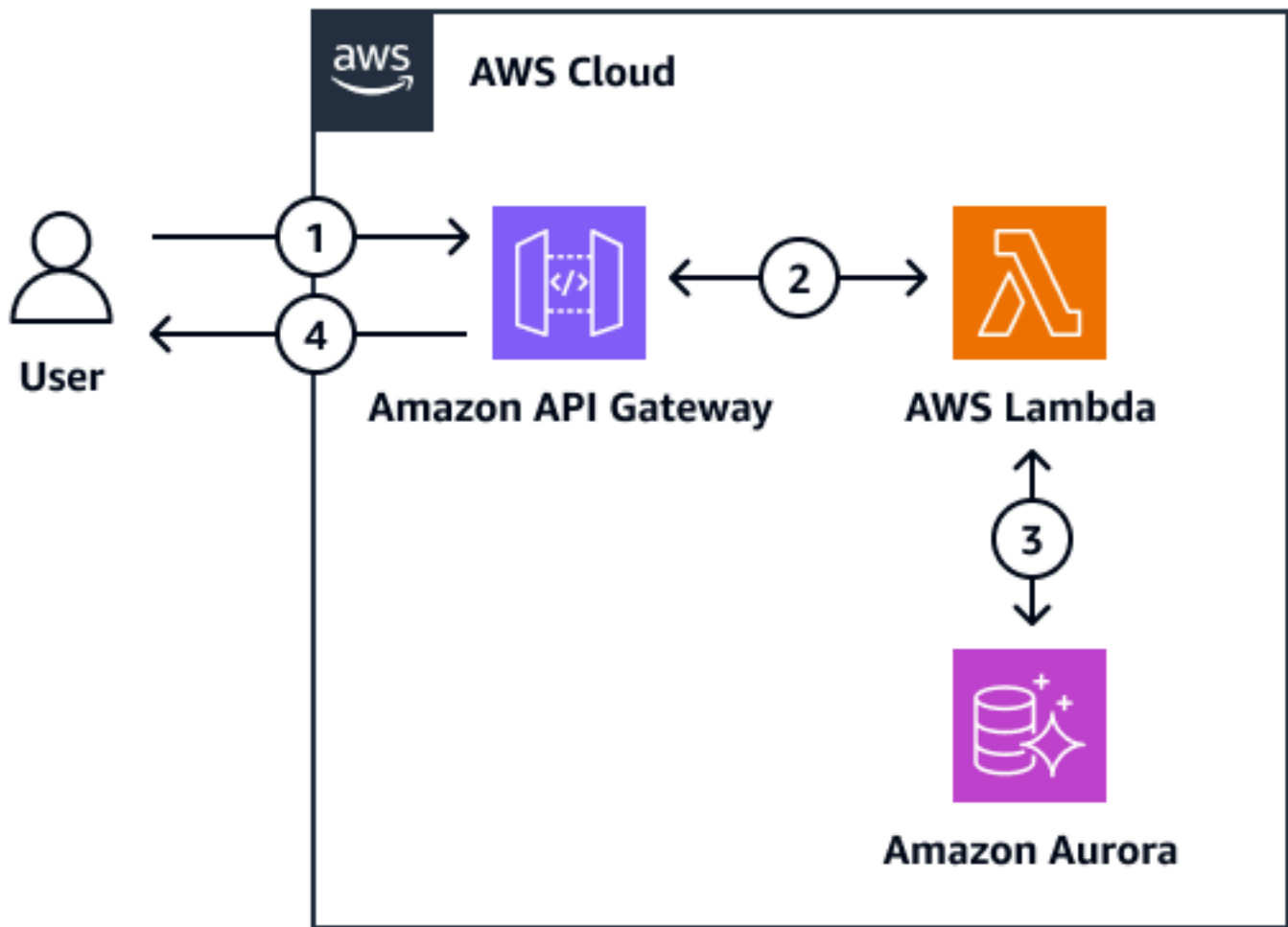
- [Dai sistemi basati sugli eventi a quelli basati sulla cognizione aumentata](#)
- [Rapido concatenamento degli schemi della saga](#)
- [Schemi di invio dinamici di routing](#)
- [Parallelizzazione e schemi di dispersione](#)
- [Schemi di orchestrazione Saga](#)
- [Evaluator Reflect-Refine Loop Patterns](#)
- [Progettazione di flussi di lavoro agentici su AWS](#)
- [Conclusioni](#)

Dai sistemi basati sugli eventi a quelli basati sulla cognizione aumentata

Le moderne architetture cloud, in particolare quelle basate su principi serverless e basate sugli eventi, si sono tradizionalmente basate su modelli come routing, fan-out e arricchimento per creare sistemi reattivi e scalabili. I sistemi di intelligenza artificiale agentica si basano su queste basi riformulandole attorno al ragionamento e alla flessibilità cognitiva. LLM-augmented Questo approccio consente funzionalità di risoluzione dei problemi e automazione più sofisticate, rivoluzionando potenzialmente il modo in cui le attività complesse vengono gestite negli ambienti cloud.

Event-driven architettura

Il diagramma seguente mostra un tipico sistema distribuito:

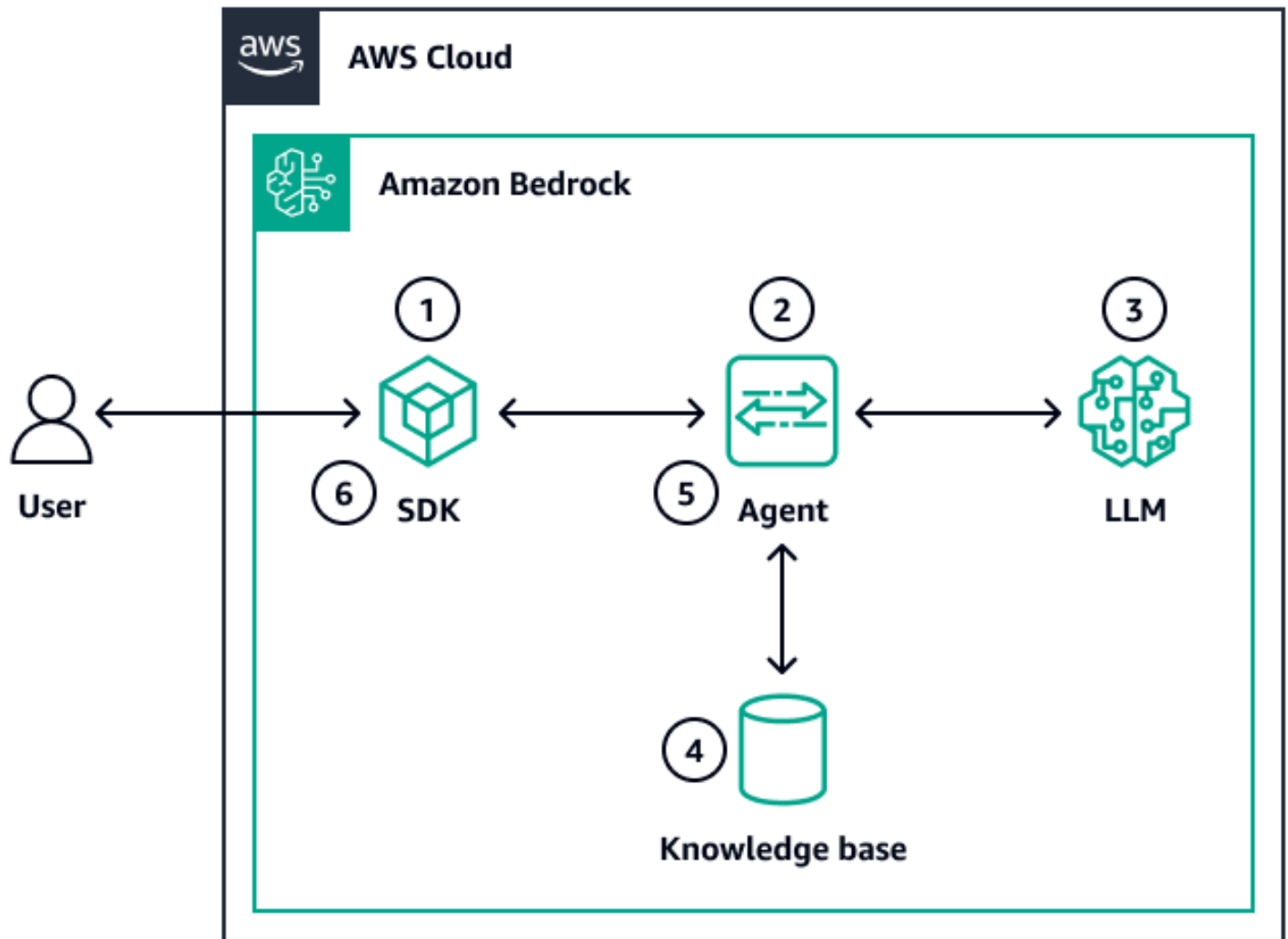


1. Un utente invia una richiesta ad Amazon API Gateway.
2. Amazon API Gateway indirizza la richiesta a una AWS Lambda funzione.
3. AWS Lambda esegue l'arricchimento dei dati interrogando un database Amazon Aurora
4. Amazon API Gateway restituisce il payload arricchito al chiamante.

Questa struttura è affidabile e scalabile, ma è fondamentalmente statica. Le regole aziendali e i percorsi logici devono essere codificati in modo esplicito e l'adattamento ai contesti mutevoli o alle informazioni incomplete è limitato.

Cognition-augmented flussi di lavoro

Le architetture agentiche aggiungono un potenziamento cognitivo a un sistema basato sugli eventi. Il diagramma seguente mostra un equivalente agentico:



1. Un utente invia una richiesta tramite una chiamata SDK o API.
2. Un agente Amazon Bedrock riceve la richiesta.
3. L'agente interpreta la query richiamando un LLM
4. L'agente esegue l'arricchimento semantico effettuando ricerche nella knowledge base di Amazon Bedrock o in altre fonti di dati esterne.
5. L'LLM sintetizza una risposta ricca di contesto e allineata agli obiettivi.
6. Il sistema restituisce una risposta sintetizzata all'utente.

In questo flusso, l'LLM utilizza la logica, comprende l'intento, recupera e combina il contesto pertinente e quindi decide il modo migliore di rispondere. Questo modello rispecchia il modello di arricchimento tradizionale, in cui i messaggi vengono arricchiti con dati esterni prima di essere

inoltrati ulteriormente. Nei sistemi agentici, tuttavia, questo arricchimento non è una ricerca statica. L'arricchimento è invece dinamico, guidato semanticamente e guidato da uno scopo.

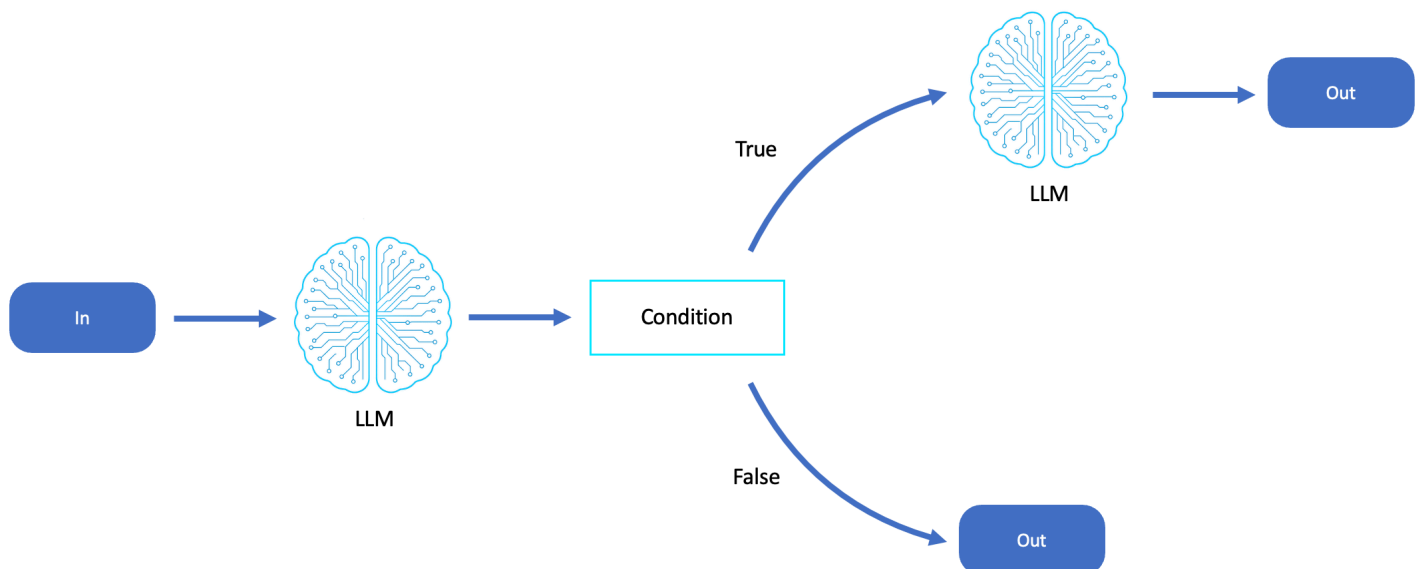
Approfondimenti fondamentali

Ogni flusso di lavoro LLM può essere mappato su un modello di flusso di lavoro agentico, che rispecchia ed evolve gli stili di architettura tradizionali basati sugli eventi. Un elemento fondamentale dei flussi di lavoro agentici è la capacità di ampliare il contesto di un LLM con dati, strumenti e memoria. Questo crea un ciclo di ragionamento informato, adattivo e allineato alle intenzioni dell'utente. Laddove i sistemi tradizionali arricchiscono i messaggi con dati di ricerca, i sistemi agentici consentono al software di agire meno come script e più come collaboratori intelligenti.

Rapido concatenamento degli schemi della saga

Reimmaginando il prompt chaining LLM come una saga basata sugli eventi, sblocciamo un nuovo modello operativo: i flussi di lavoro diventano distribuiti, recuperabili e coordinati semanticamente tra agenti autonomi. Ogni fase di pronta risposta viene riformulata come un'attività atomica, emessa come evento, consumata da un agente dedicato e arricchita con metadati contestuali.

Il diagramma seguente è un esempio di concatenamento di prompt LLM:

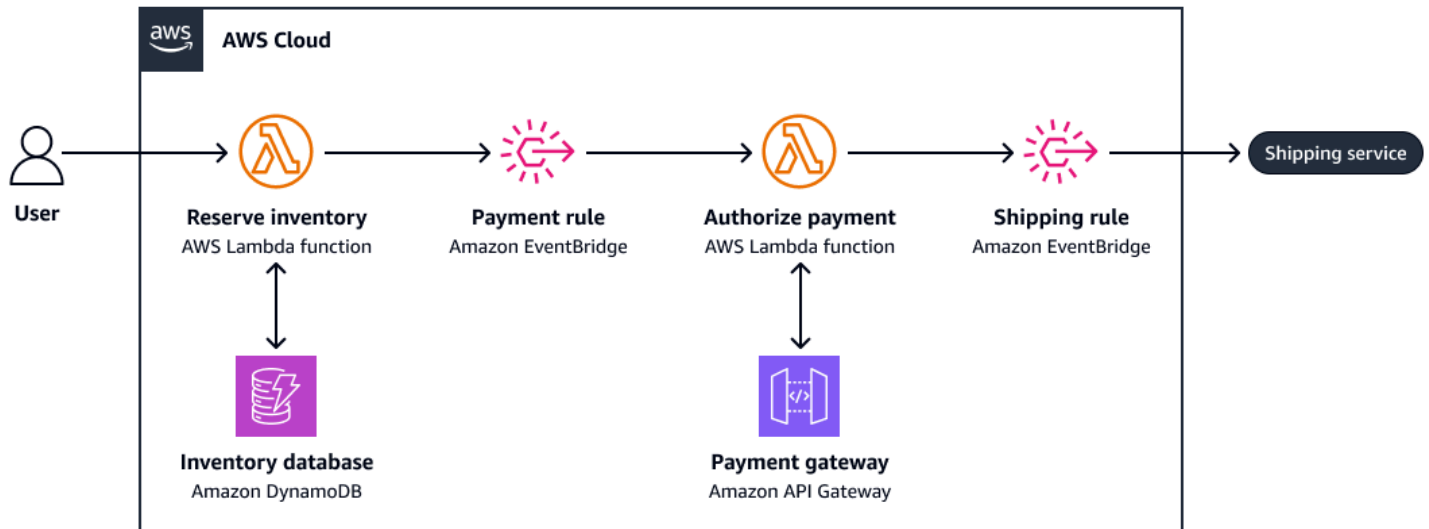


Coreografia saga

Il modello coreografico della saga è un approccio di implementazione in sistemi distribuiti che non ha un coordinatore centrale. Invece, ogni servizio o componente pubblica eventi che attivano l'azione

successiva del flusso di lavoro. Questo modello è ampiamente utilizzato nei sistemi distribuiti per la gestione delle transazioni tra più servizi. In una saga, il sistema esegue una serie di transazioni locali coordinate. Se una fallisce, il sistema attiva azioni di compensazione per mantenere la coerenza.

Il diagramma seguente è un esempio di coreografia da saga:



1. Inventario di riserva
2. Autorizza il pagamento
3. Crea un ordine di spedizione

Se la fase 3 fallisce, il sistema richiama azioni di compensazione (ad esempio, annullare un pagamento o rilasciare l'inventario).

Questo modello è particolarmente utile nelle architetture basate sugli eventi, in cui i servizi sono collegati in modo flessibile e gli stati devono essere risolti in modo coerente nel tempo, anche in presenza di guasti parziali.

Schema di concatenamento rapido

Il concatenamento rapido ricorda lo schema della saga sia nella struttura che nello scopo. Esegue una serie di passaggi di ragionamento che vengono creati in sequenza preservando il contesto e consentendo rollback e revisioni.

Coreografia dell'agente

1. LLM interpreta una query utente complessa e genera un'ipotesi

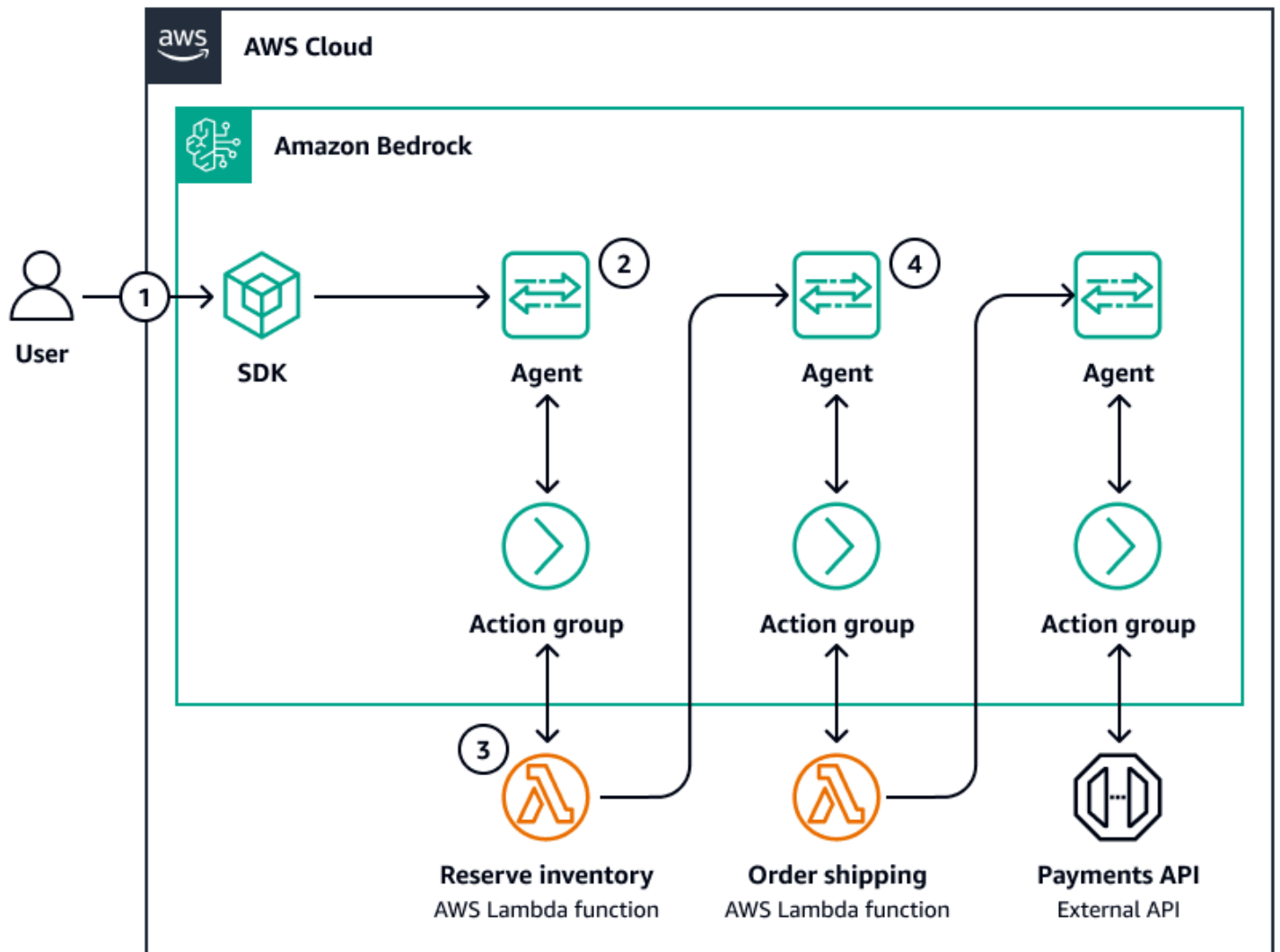
2. LLM elabora un piano per risolvere il problema
3. LLM esegue una sottoattività (ad esempio, utilizzando una chiamata a uno strumento o recuperando conoscenze)
4. LLM perfeziona l'output o rivisita un passaggio precedente se ritiene che un risultato non sia soddisfacente

Se un risultato intermedio è difettoso, il sistema può eseguire una delle seguenti operazioni:

- Riprova i passaggi utilizzando un approccio diverso
- Torna a un prompt precedente e ripianifica
- Utilizzate un ciclo di valutazione (ad esempio, dal pattern valutatore-ottimizzatore) per rilevare e correggere gli errori

Analogamente al modello saga, il prompt chaining consente meccanismi di avanzamento e rollback parziali. Ciò avviene attraverso il perfezionamento e la LLM-directed correzione iterativi anziché attraverso la compensazione delle transazioni del database.

Il diagramma seguente è un esempio di coreografia tra agenti:



1. Un utente invia una richiesta tramite un SDK.
2. Un agente Amazon Bedrock orchestra il ragionamento attraverso quanto segue:
 - Interpretazione (LLM)
 - Pianificazione (LLM)
 - Esecuzione tramite uno strumento o una base di conoscenze
 - Costruzione della risposta
3. Se uno strumento si guasta o restituisce dati insufficienti, l'agente può ripianificare o riformulare dinamicamente l'attività.
4. La memoria (ad esempio, un archivio vettoriale a breve termine) può preservare il proprio stato attraverso i passaggi

Da asporto

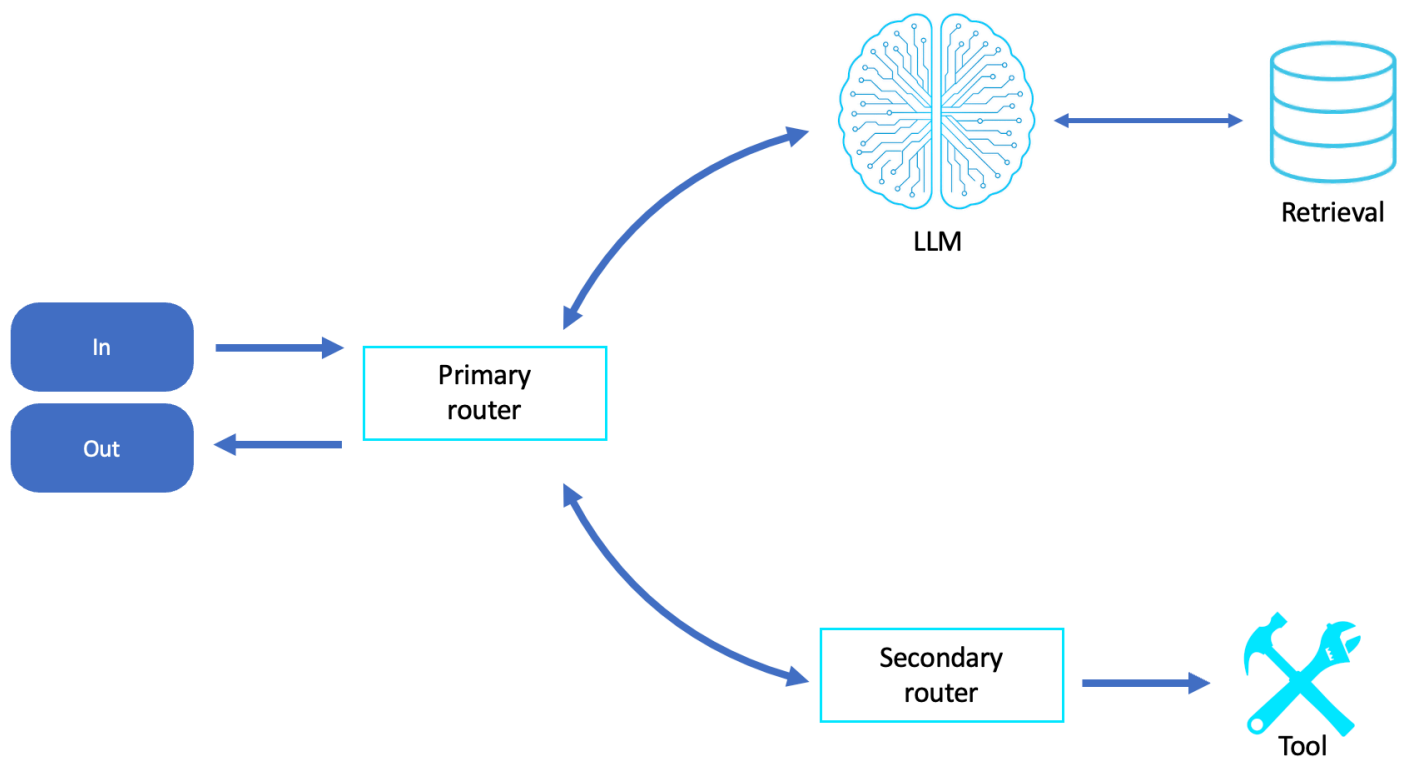
Laddove Saga Pattern gestisce le chiamate di servizio distribuite con una logica di compensazione, il prompt chaining gestisce le attività di ragionamento con sequenziamento riflessivo e ripianificazione adattiva. Entrambi i sistemi consentono progressi incrementali, punti decisionali decentralizzati e ripristino degli errori, e tutto ciò avviene attraverso un ragionamento informato anziché un rigido rollback.

Il concatenamento rapido introduce il ragionamento transazionale, che è l'equivalente cognitivo delle saghe. Cioè, ogni «pensiero» viene rivalutato, rivisto o abbandonato come parte di un dialogo più ampio mirato.

Schemi di invio dinamici di routing

Nei moderni sistemi agentici, in cui le attività spaziano dall'analisi dei documenti alla generazione autonoma di software, la capacità di indirizzare dinamicamente le richieste all'agente o modello di linguaggio di grandi dimensioni (LLM) più capace diventa fondamentale. La logica di routing statica, spesso incorporata negli script di orchestrazione o nei livelli API, non ha l'adattabilità richiesta per ambienti in tempo reale, multimodello e con più capacità. Per risolvere questo problema, i flussi di lavoro di routing LLM possono essere trasformati in un'architettura basata sugli eventi che sfrutta uno schema di invio dinamico, trasformando le chiamate LLM in eventi indirizzati in modo intelligente e sensibili al contesto.

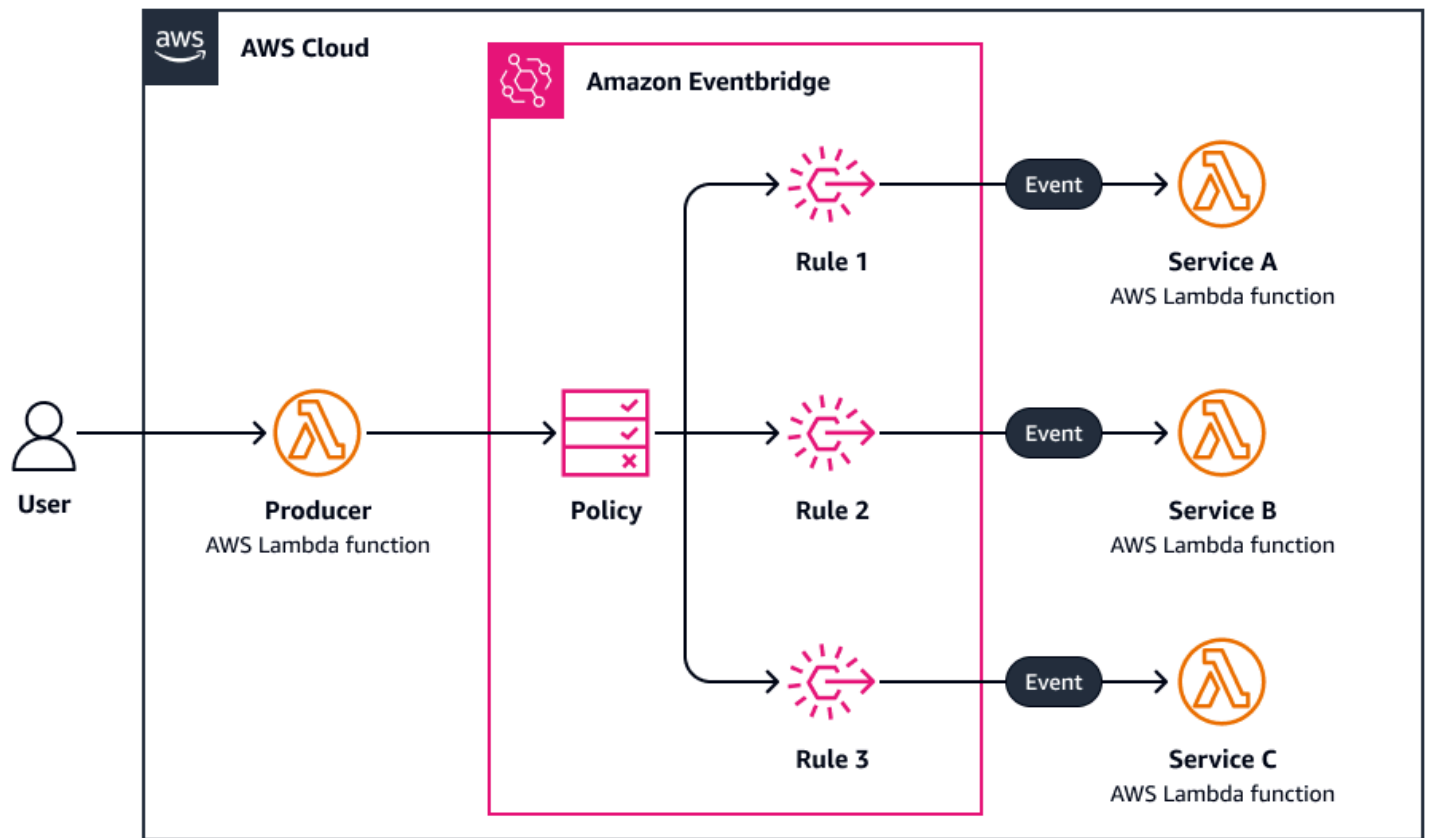
Il diagramma seguente è un esempio di routing LLM:



Invio dinamico

Nei sistemi distribuiti tradizionali, il pattern di invio dinamico seleziona e richiama servizi specifici in fase di esecuzione in base agli attributi degli eventi in entrata, come il tipo di evento, l'origine e il payload. Questo viene comunemente implementato utilizzando Amazon EventBridge, che può valutare e indirizzare gli eventi in entrata verso destinazioni appropriate (ad esempio AWS Step Functions, AWS Lambda funzioni o attività di Amazon Elastic Container Service).

Il diagramma seguente è un esempio di invio dinamico:



1. Un'applicazione emette un evento (ad esempio, `{"type": «orderCreated», «priority»: «high»}`).
2. Amazon EventBridge valuta l'evento in base alle sue regole di routing.
3. In base agli attributi di un evento, il sistema invia dinamicamente a quanto segue:
 - `HighPriorityOrderProcessor(servizio A)`
 - `StandardOrderProcessor(servizio B)`
 - `UpdateOrderProcessor(servizio C)`

Questo modello supporta l'accoppiamento libero, la specializzazione basata sul dominio e l'estensibilità del runtime. Ciò consente ai sistemi di rispondere in modo intelligente ai mutevoli requisiti e alla semantica degli eventi.

LLM-based routing

Nei sistemi agentici, il routing esegue anche la delega dinamica delle attività, ma invece delle EventBridge regole di Amazon o dei filtri dei metadati, l'LLM classifica e interpreta l'intento dell'utente

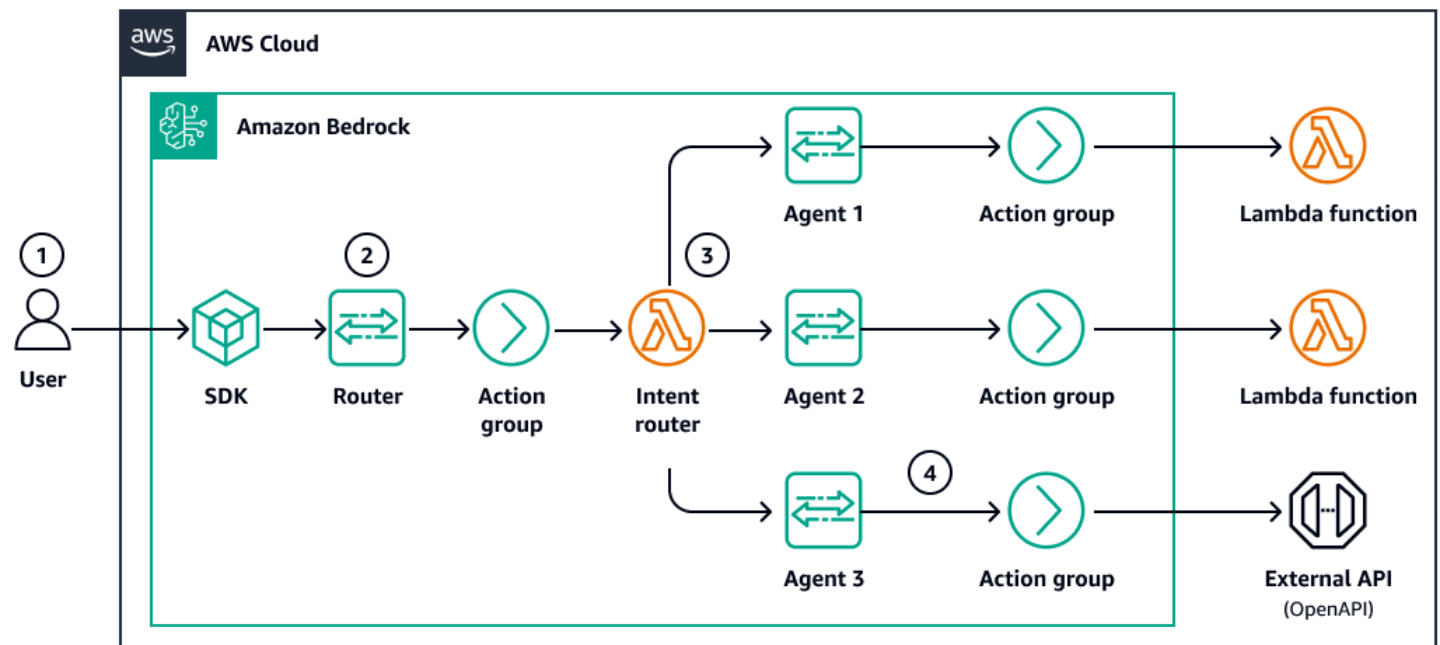
attraverso il linguaggio naturale. Il risultato è una forma di dispacciamento flessibile, semantica e adattiva.

Router agente

Questa architettura consente un ampio invio basato sugli intenti senza schemi o tipi di eventi predefiniti, ideale per input non strutturati e query complesse.

1. Un utente invia la richiesta «Potete aiutarmi a rivedere le condizioni del mio contratto?»
2. L'LLM lo interpreta come un'attività documentale legale.
3. L'agente indirizza l'attività a uno o più dei seguenti:
 - Modello di richiesta di revisione del contratto
 - Subagente per il ragionamento legale
 - Strumento di analisi dei documenti

Il diagramma seguente è un esempio di router agente:



1. Un utente invia una richiesta in linguaggio naturale tramite un SDK.
2. Un agente Amazon Bedrock utilizza un LLM per classificare l'attività (ad esempio legale, tecnica o di pianificazione).

3. L'agente indirizza dinamicamente l'attività attraverso un gruppo di azioni per richiamare l'agente richiesto:
 - Domain-specific agente
 - Catena di utensili specializzata
 - Configurazione personalizzata del prompt
4. Il gestore selezionato elabora l'operazione e restituisce una risposta personalizzata.

Da asporto

Laddove la distribuzione dinamica tradizionale utilizza le EventBridge regole di Amazon per il routing basate su attributi di eventi strutturati, il routing agentic utilizza gli LLM per classificare e indirizzare semanticamente le attività in base al significato e all'intento. Ciò amplia la flessibilità del sistema abilitando quanto segue:

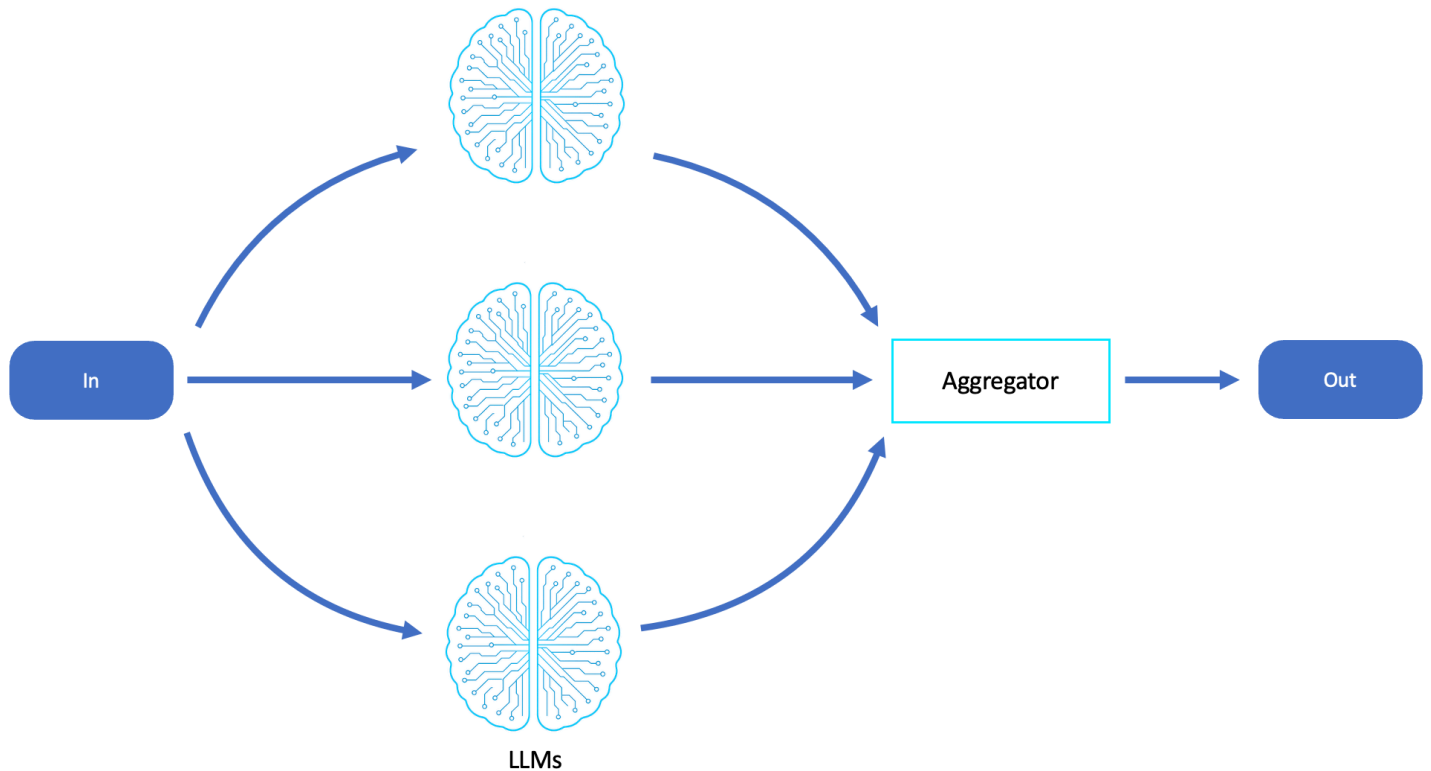
- Comprensione più ampia degli input
- Fallback intelligente e selezione degli strumenti
- Estensibilità naturale attraverso nuovi ruoli di agente o stili di prompt

L'agentic routing sostituisce le regole rigide con il dispatching cognitivo dinamico, che consente ai sistemi di evolversi con il linguaggio anziché con il codice.

Parallelizzazione e schemi di dispersione

Molte attività avanzate di ragionamento e generazione, come il riepilogo di documenti di grandi dimensioni, la valutazione di più percorsi di soluzione o il confronto di diverse prospettive, traggono vantaggio dall'esecuzione parallela dei prompt. I flussi di lavoro sequenziali tradizionali non sono all'altezza quando sono richieste scalabilità, reattività e tolleranza agli errori. Per ovviare a questo problema, è possibile reinventare la LLM-based parallelizzazione utilizzando un modello di scatter-gather basato sugli eventi, in cui le attività vengono assegnate dinamicamente ad agenti autonomi e i risultati sintetizzati in modo intelligente.

Il diagramma seguente è un esempio di flusso di lavoro di parallelizzazione LLM:



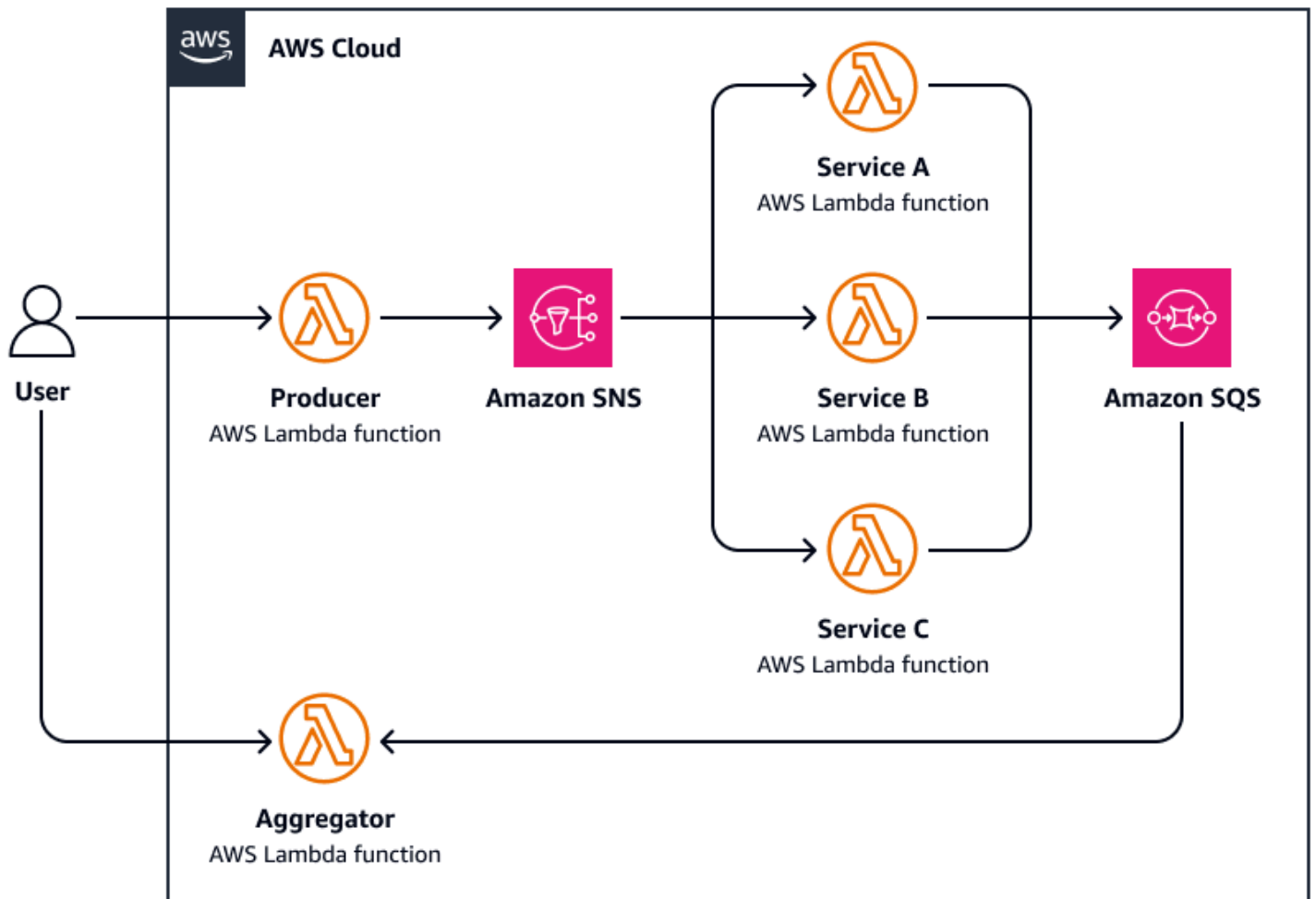
Scatter-gather

Nei sistemi distribuiti, un pattern scatter-gather invia le attività a più servizi o unità di elaborazione in parallelo, attende le loro risposte e quindi aggrega i risultati in un output consolidato. A differenza del fan-out, lo scatter-gather è coordinato perché prevede risposte e di solito applica la logica per combinare, confrontare e selezionare i risultati.

Le implementazioni comuni per la parallelizzazione e lo scatter-gather includono quanto segue:

- AWS Step Functions mappare uno stato per l'esecuzione di attività parallele
- AWS Lambda con concorrenza, coordinando i risultati di più funzioni richiamate
- Amazon EventBridge con ID di correlazione e flussi di lavoro di aggregazione
- Schema di controller personalizzato per gestire il fan-out e raccogliere risultati utilizzando Amazon Simple Storage Service (Amazon S3), Amazon DynamoDB o code

Il diagramma seguente è un esempio di scatter-gather:



1. Un utente invia una richiesta a una funzione di coordinamento centrale che suddivide l'attività pubblicando messaggi paralleli su un argomento di Amazon Simple Notification Service (Amazon SNS).
2. Ogni messaggio include i metadati delle attività e viene indirizzato a un operatore specializzato. AWS Lambda
3. Ogni lavoratore elabora AWS Lambda in modo indipendente la sottoattività assegnata (ad esempio, interrogazione di un'API esterna, elaborazione di un documento e analisi dei dati).
4. I risultati vengono scritti su un livello di storage comune, come Amazon Simple Queue Service (Amazon SQS).
5. La funzione di aggregazione attende il completamento di tutte le risposte, quindi esegue le seguenti operazioni:
 - Raccoglie e aggrega i risultati (ad esempio, unisce i riepiloghi, seleziona le migliori corrispondenze)

- Invia una risposta finale o attiva un flusso di lavoro a valle

I casi d'uso più comuni dei pattern scatter-gather includono i seguenti:

- Ricerca federata
- Motori di confronto dei prezzi
- Analisi aggregata dei dati
- Inferenza multimodello

LLM-based parallelizzazione (cognizione scatter-gather)

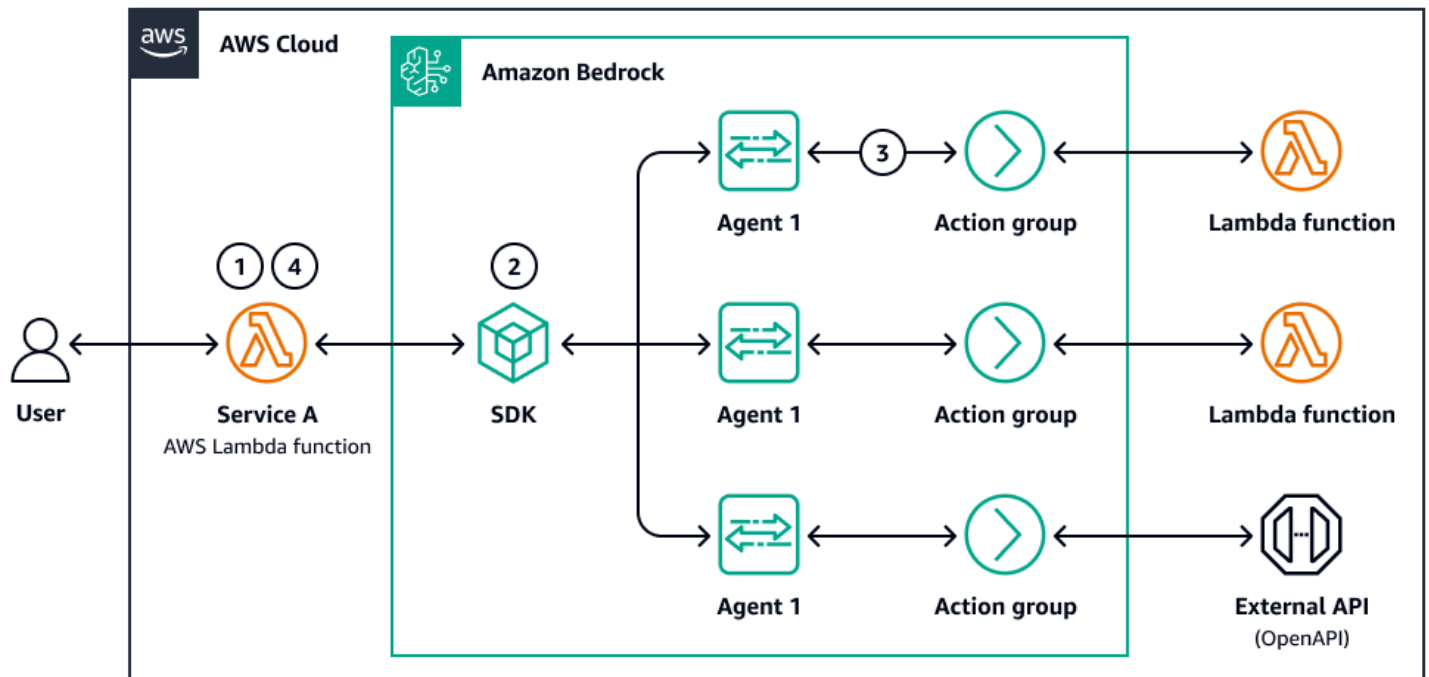
Nei sistemi agentici, la parallelizzazione rispecchia da vicino lo scatter-gather distribuendo le sottoattività su più chiamate o agenti LLM, ciascuno dei quali ragiona in modo indipendente su una parte del problema. I risultati restituiti vengono raccolti e sintetizzati da un processo di aggregazione, che spesso è un altro LLM o un altro agente di controllo.

Parallelizzazione degli agenti

1. Un agente invia una richiesta «Riepiloga le informazioni raccolte in questi 10 report».
2. Suddivide i report in 10 attività di riepilogo LLM parallele.
3. Quando restituisce tutti i riepiloghi, l'agente esegue le seguenti operazioni:
 - Aggrega i riepiloghi in un briefing unificato
 - Identifica temi o contraddizioni
 - Invia l'output sintetizzato all'utente

Questo flusso di lavoro agentico consente un ragionamento parallelo scalabile, modulare e adattivo. È ideale per i casi d'uso che richiedono un throughput cognitivo elevato.

Il diagramma seguente è un esempio di parallelizzazione degli agenti:



1. Un utente invia una query o un set di documenti composto da più parti.
2. Un controller AWS Lambda o una funzione Step distribuisce le sottoattività. Ogni attività richiama una chiamata o un subagente Amazon Bedrock LLM con il proprio prompt.
3. Quando le chiamate e le sottoattività sono complete, i risultati vengono archiviati (ad esempio, in Amazon S3 o nell'archivio di memoria) e una fase di aggregazione unisce, confronta o filtra gli output.
4. Il sistema restituisce la risposta finale all'utente o all'agente downstream.

Questo sistema dispone di un ciclo di ragionamento distribuito con tracciabilità, tolleranza ai guasti e logica opzionale di ponderazione o selezione dei risultati.

Da asporto

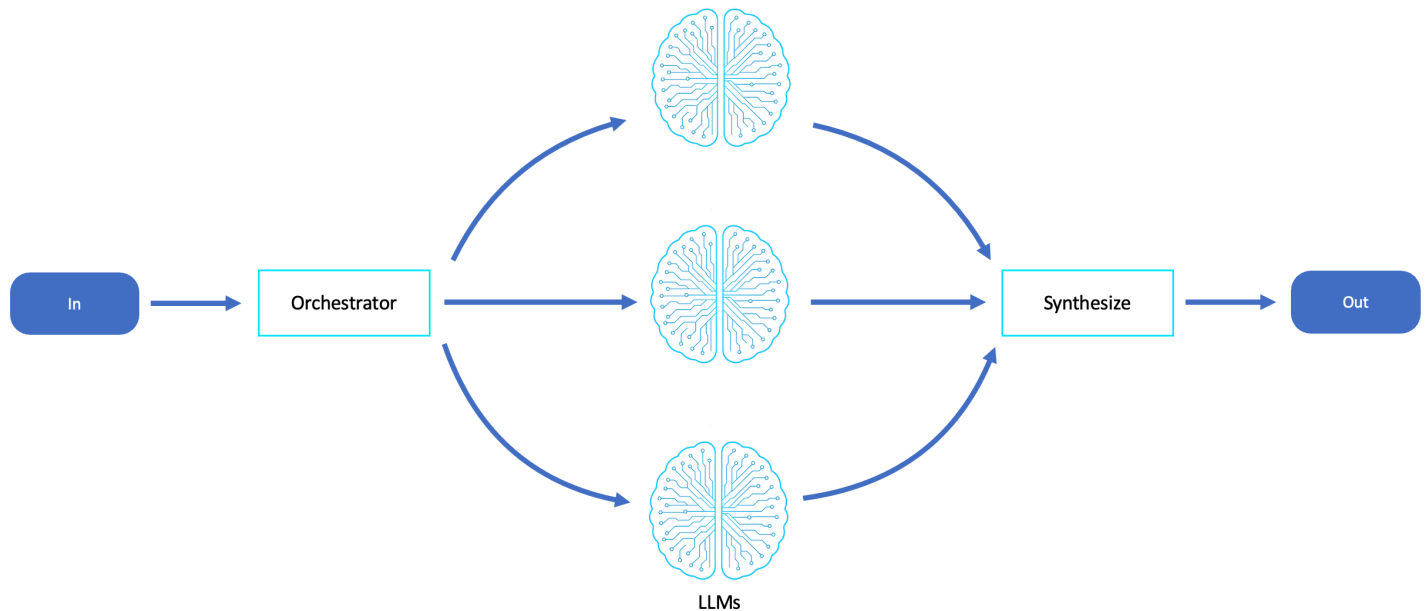
La parallelizzazione agentica utilizza modelli scatter-gather per distribuire le attività LLM, abilitando l'elaborazione parallela e la sintesi intelligente dei risultati.

Schemi di orchestrazione Saga

Man mano che i flussi di lavoro guidati dai LLM diventano sempre più complessi e comprendono catene di richieste, fasi di elaborazione dei dati, invocazioni di strumenti e collaborazione tra agenti,

la necessità di un'orchestrazione intelligente diventa essenziale. Aniché fare affidamento su script strettamente associati o flussi di esecuzione statici predeterminati, questi flussi di lavoro possono essere implementati come modelli di orchestrazione basati sugli eventi, consentendo ai sistemi di coordinare, monitorare e adattare dinamicamente le attività in più fasi tra agenti autonomi. LLM-based

Il diagramma seguente è un esempio di orchestratore:



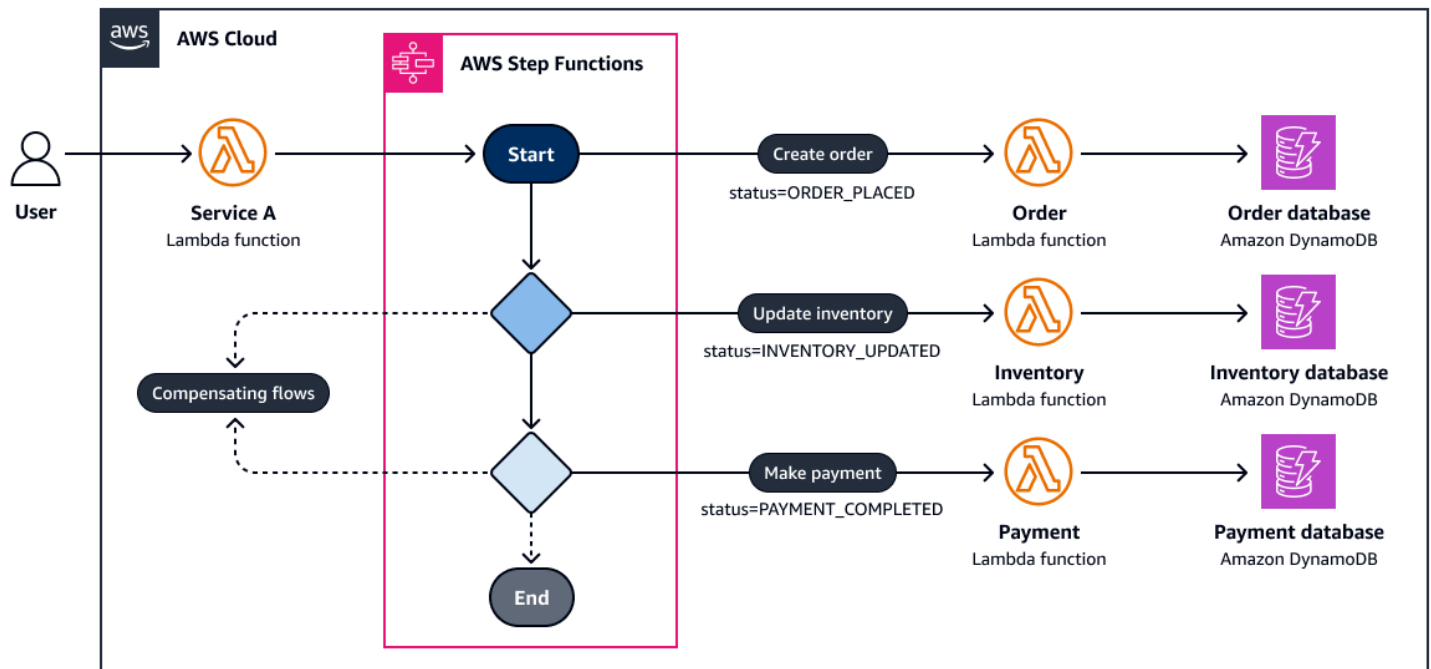
Orchestrazione di eventi

Nei sistemi distribuiti tradizionali, l'orchestrazione degli eventi si riferisce a uno schema in cui un coordinatore centrale gestisce un flusso di lavoro complesso indirizzando esplicitamente il flusso di controllo su più servizi o attività. A differenza della coreografia degli eventi (in cui ogni servizio reagisce in modo indipendente), l'orchestrazione fornisce logica, visibilità e controllo centralizzati sull'intero processo.

Questa operazione viene in genere implementata utilizzando i seguenti strumenti:

- AWS Step Functions— Definizione ed esecuzione di flussi di lavoro basati sullo stato
- AWS Lambda— Esegui attività discrete all'interno del flusso orchestrato
- Amazon SQS o Amazon EventBridge: attiva passaggi o risposte asincroni

Il diagramma seguente è un esempio di orchestrazione di saga:



Un AWS Step Functions flusso di lavoro gestisce un processo di ordinazione dei clienti:

1. Crea ordine (AWS Lambda)
2. Aggiorna l'inventario (AWS Lambda)
3. Effettua il pagamento (AWS Lambda)

L'orchestratore coordina ogni fase gestendo nuovi tentativi, branch paralleli, timeout e guasti.

Role-based sistema di agenti (orchestratore)

Nei sistemi agentici, il pattern di orchestrazione rispecchia l'orchestrazione degli eventi ma distribuisce la logica tra più agenti di ragionamento, ciascuno con un ruolo o una specializzazione definiti. Un agente di orchestrazione centrale interpreta l'attività complessiva, la scompone in sottoattività e le delega agli agenti di lavoro, ciascuno ottimizzato per un particolare dominio (ad esempio, ricerca, codifica, riepilogo, revisione).

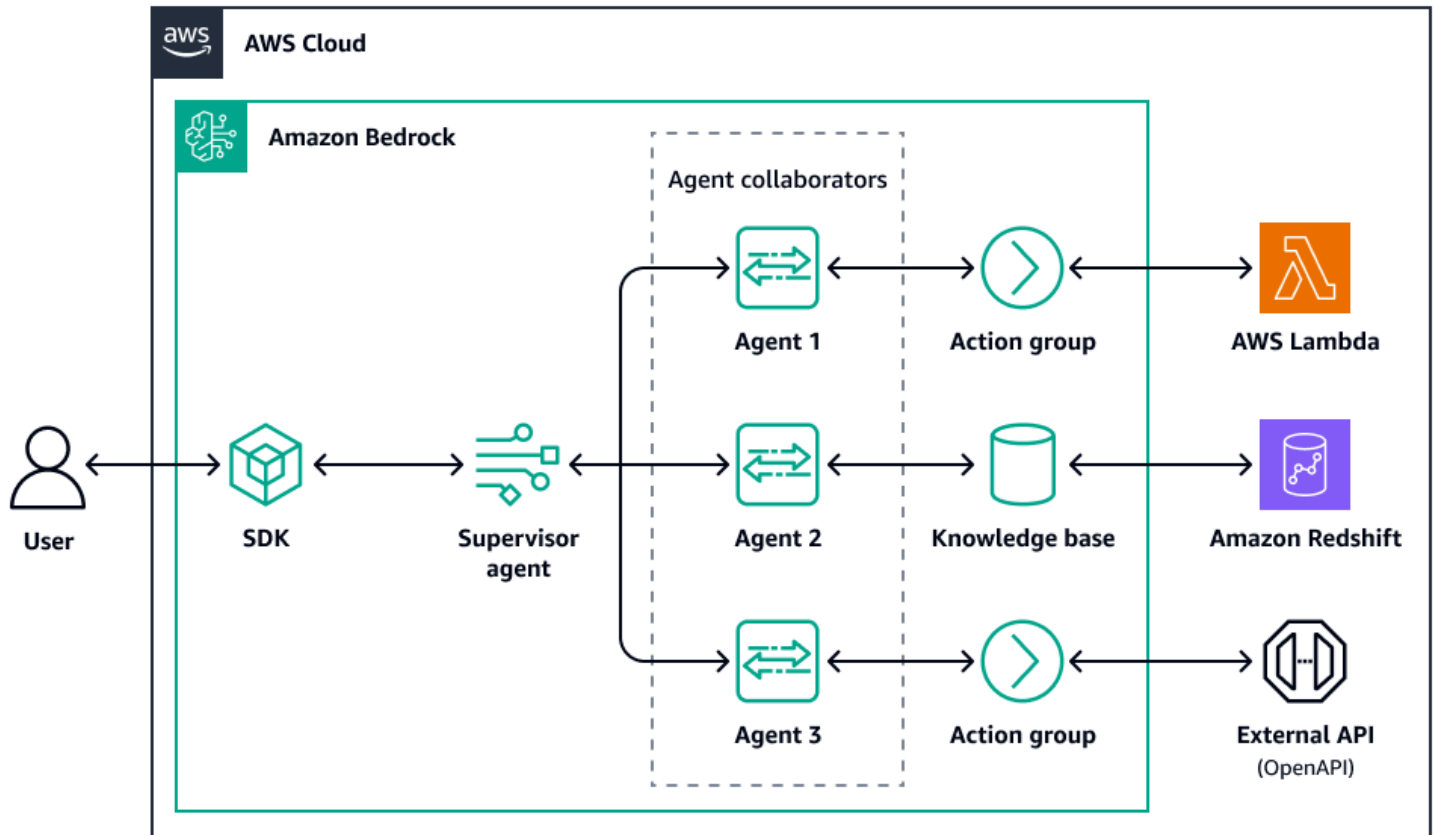
Supervisore

1. Un utente invia la domanda «Crea un brief del progetto e riepiloga i primi 5 concorrenti».
2. L'agente orchestrator esegue le seguenti operazioni:
 - Assegna a un agente di ricerca il compito di trovare i dati dei concorrenti

- Invia i risultati non elaborati a un agente di riepilogo
- Trasmette i risultati a un agente addetto alla redazione di brief
- Compila l'output finale per l'utente

Ogni agente opera in modo indipendente, ma l'orchestratore coordina le attività. È come una funzione Lambda che gestisce le attività del flusso di lavoro.

Il diagramma seguente mostra un esempio di supervisore:



1. Un utente invia un'attività a un agente supervisore di Amazon Bedrock.
2. L'agente supervisore analizza la richiesta in sottoattività per ogni collaboratore agente.
3. Ogni sottoattività viene assegnata a un agente collaboratore con istruzioni o toolchain specifici per il ruolo.
4. Gli agenti di lavoro chiamano API o strumenti esterni tramite un gruppo di azione.
5. Ogni agente di lavoro restituisce l'output in un formato strutturato.
6. Quando tutti i lavoratori restituiscono i risultati, il supervisore valuta, sintetizza e restituisce la risposta finale.

Questa struttura consente modularità, adattabilità e introspezione in flussi di lavoro complessi con agenti in più fasi.

Da asporto

Laddove l'orchestrazione degli eventi utilizza il controllo centralizzato (ad esempio AWS Step Functions) per dirigere l'esecuzione del servizio, i sistemi di agenti basati sui ruoli utilizzano un agente di LLM-powered orchestrazione per ragionare sull'obiettivo, delegare le attività secondarie agli agenti di lavoro e sintetizzare l'output finale.

In entrambi i paradigmi, l'orchestratore esegue le seguenti operazioni:

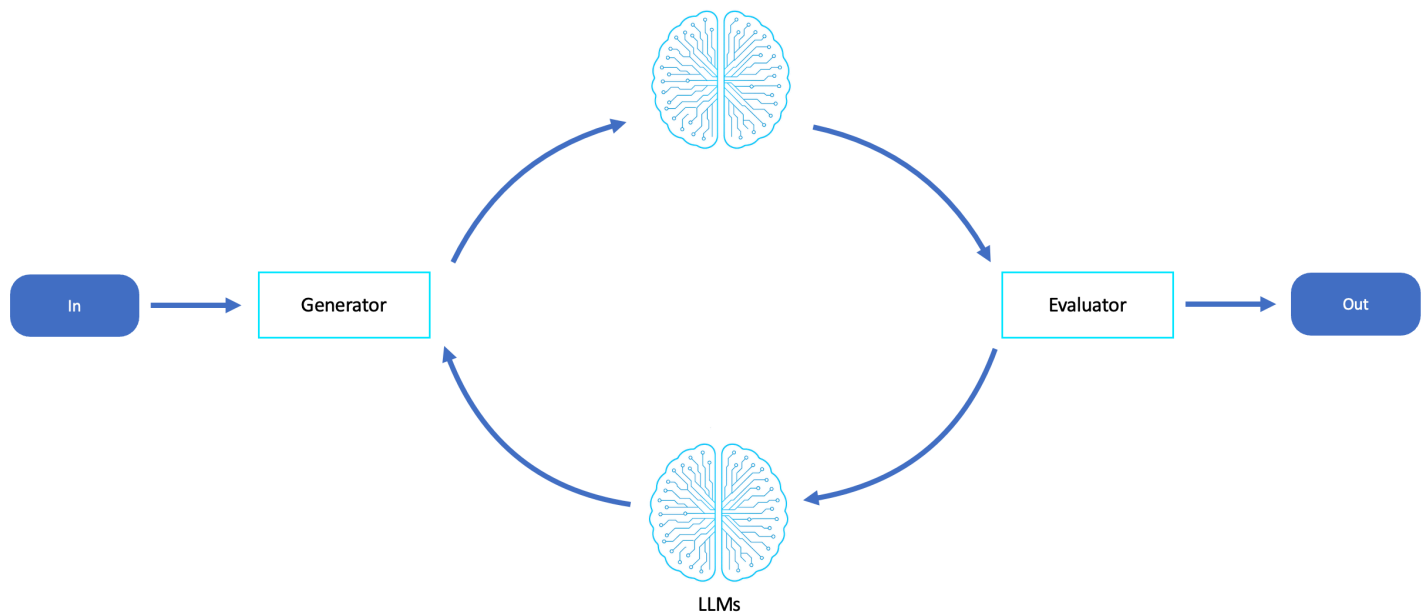
- Mantiene il contesto e il flusso di esecuzione
- Gestisce la ramificazione, il sequenziamento e la gestione degli errori
- Produce un risultato unificato a partire da componenti distribuiti

L'orchestrazione agentica, tuttavia, aggiunge ragionamento, adattabilità e delega semantica. Ciò lo rende adatto a compiti aperti, ambigui e in continua evoluzione.

Evaluator Reflect-Refine Loop Patterns

Attività come la generazione di codice, il riepilogo o il processo decisionale autonomo traggono grandi vantaggi dal feedback in fase di esecuzione, che consente al sistema di evolversi attraverso l'osservazione e il perfezionamento. Per renderlo operativo, il ciclo reflect-refine può essere implementato come un circuito di controllo del feedback basato sugli eventi, un modello ispirato all'ingegneria dei sistemi, adattato per flussi di lavoro autonomi e intelligenti.

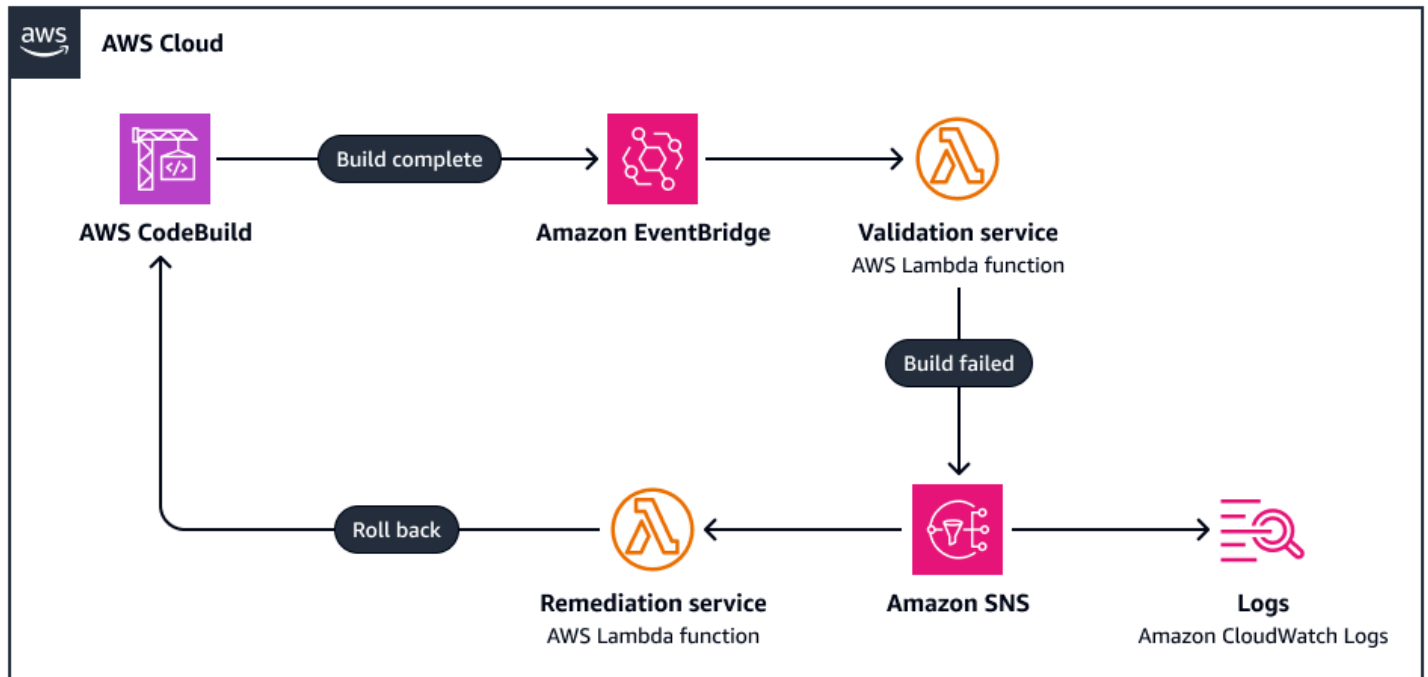
Il diagramma seguente è un esempio di ciclo di feedback reflect-refine di Evaluator:



Circuito di controllo del feedback

Un circuito di controllo del feedback è uno schema che monitora i propri output e comportamenti, li valuta in base a criteri definiti o a uno stato desiderato e quindi regola le proprie azioni di conseguenza. Questa architettura si ispira alla teoria del controllo ed è fondamentale in domini come le pipeline di automazione, integrazione continua e distribuzione continua (CI/CD) e le operazioni di apprendimento automatico.

Il diagramma seguente è un esempio di circuito di controllo del feedback:



1. Una pipeline di distribuzione emette un evento BuildComplete.
2. L'evento attiva un processo di test o valutazione automatizzato che convalida la build.
3. Se la convalida fallisce (ad esempio, a causa di test non riusciti, problemi di sicurezza o violazione delle politiche), il sistema:
 - Emette un evento BuildComplete
 - Registra il problema o invia una notifica
 - Attiva un'azione riparativa o correttiva, ad esempio il rollback, l'applicazione di una patch o un nuovo tentativo

Il ciclo continua fino a quando non produce un risultato o un'escalation accettabile o fino a quando non si verifica un timeout. Questo modello viene comunemente utilizzato per quanto segue:

- EventBridge Regole di Amazon per indirizzare gli eventi alle attività di valutazione o riparazione
- AWS Step Functions per una logica di ripetizione iterativa e una ramificazione dei risultati della valutazione
- Amazon Simple Notification Service (Amazon SNS) o allarmi Amazon per i trigger di CloudWatch feedback e gli avvisi
- AWS Lambda funzioni o lavoratori containerizzati per applicare azioni correttive

Circuito di controllo del feedback (valutatore)

Il flusso di lavoro di un valutatore è un ciclo di feedback cognitivo basato su LLM o agenti di ragionamento. Il processo è costituito dai seguenti elementi:

1. Un agente generatore o LLM produce un output (ad esempio, un piano, una risposta o una bozza).
2. Un agente valutatore esamina il risultato utilizzando un prompt di critica o una rubrica di valutazione.
3. In base al feedback, l'agente originale o un nuovo agente di ottimizzazione rivede l'output.

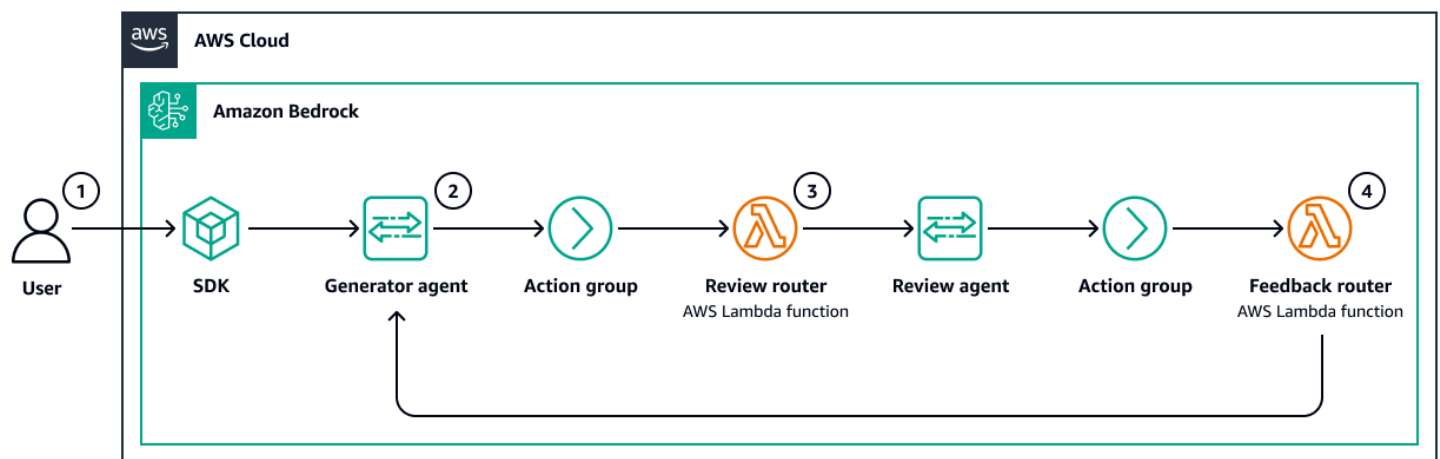
Il ciclo si ripete finché il risultato non soddisfa una serie di criteri, non viene approvato o raggiunge un limite di tentativi.

Valutatore

1. Un utente chiede a un agente di scrivere un riepilogo delle politiche.
2. L'agente generatore lo redige.
3. Un agente valutatore verifica la copertura, il tono e la correttezza legale.
4. Se la risposta è inadeguata, viene perfezionata e inviata nuovamente fino alla convergenza del ciclo di feedback.

Ciò consente l'autovalutazione, il perfezionamento iterativo e il controllo adattivo dell'output, il tutto senza l'intervento umano.

Il diagramma seguente è un esempio di circuito di controllo del feedback (valutatore):



1. Un utente esegue un'attività (ad esempio, redigere una strategia aziendale).
2. Un agente Amazon Bedrock genera una bozza iniziale utilizzando un LLM.
3. Un secondo agente (o un prompt di follow-up) esegue una valutazione strutturata (ad esempio, «valuta questo risultato in base a chiarezza, completezza e tono»).
4. Se la valutazione scende al di sotto di una soglia, la risposta viene modificata da:
 - Reinvocare il generatore con una critica incorporata
 - Invio del feedback a un agente di raffinazione specializzato
 - Iterazione fino al raggiungimento di una risposta accettabile

Componenti opzionali come AWS Lambda e controller AWS Step Functions possono gestire soglie di feedback, nuovi tentativi e strategie di fallback.

Cose da asporto

Laddove i tradizionali circuiti di controllo del feedback utilizzano eventi, metriche e logica di correzione per convalidare e regolare il comportamento del sistema, i loop di valutazione agentica utilizzano agenti di ragionamento per valutare, riflettere e rivedere l'output in modo dinamico.

In entrambi i paradigmi:

- L'output viene valutato dopo essere stato generato
- Le azioni correttive o di perfezionamento vengono attivate in base al feedback
- Il sistema si adatta continuamente alla qualità o all'obiettivo prefissato

La versione agentica trasforma la convalida statica in riflessione semantica, permettendo agli agenti di automiglioramento di valutare la propria efficacia.

Progettazione di flussi di lavoro agentici su AWS

Ogni modello in questa guida può essere creato utilizzando i servizi AWS. Gli agenti Amazon Bedrock forniscono canali di orchestrazione, accesso ai dati e interazione.

Componente

Servizio AWS

Scopo

Ragionamento LLM	Amazon Bedrock	Logica degli agenti, pianificazione, utilizzo degli strumenti
Esecuzione degli strumenti	AWS Lambda, Amazon ECS, Amazon SageMaker	Ospita strumenti esterni per agenti
Memoria e RAG	Base di conoscenze Amazon Bedrock, Amazon S3, OpenSearch	Memoria persistente e semantica
Orchestrazione	AWS Step Functions	Coordinamento di attività e agenti in più fasi
Routing degli eventi	Amazon EventBridge, Amazon SQS	Messaggistica interagente disaccoppiata
Interfaccia utente	Amazon API Gateway AWS AppSync, SDK	Punti di ingresso per applicazioni o sistemi
Monitoraggio	Amazon CloudWatch AWS X-Ray, AWS Distro per OpenTelemetry	Osservabilità e introspezione degli agenti

Conclusioni

I modelli di flusso di lavoro agentici sono la prossima fase evolutiva delle architetture basate sugli eventi, in cui la logica aziendale non è definita staticamente ma ragionata dinamicamente attraverso l'utilizzo di una cognizione migliorata del modello di linguaggio (LLM). Combinando le tradizionali primitive native del cloud con flussi di lavoro LLM e modelli di progettazione degli agenti, le organizzazioni possono creare sistemi adattivi, intelligenti e modulari che rispondono con uno scopo e imparano dall'esperienza.

In questi modelli, Amazon Bedrock è la porta di accesso alla cognizione agentica, che consente LLM-based agli agenti di accedere ai flussi di lavoro degli eventi, interagire con strumenti e memoria e fornire risultati strutturati, tracciabili e allineati.

Durante la progettazione e l'implementazione di sistemi agentici, questi modelli di flusso di lavoro forniscono modelli per la creazione di architetture AI autonome e componibili. Questi sistemi si basano su best practice serverless e potenziati con modelli di base intelligenti.

Cronologia dei documenti

La tabella seguente descrive le modifiche significative apportate a questa guida. Per ricevere notifiche sugli aggiornamenti futuri, puoi abbonarti a un [feed RSS](#).

Modifica	Descrizione	Data
Pubblicazione iniziale	—	14 luglio 2025

Le traduzioni sono generate tramite traduzione automatica. In caso di conflitto tra il contenuto di una traduzione e la versione originale in Inglese, quest'ultima prevarrà.