



Pola dan alur kerja Agentic AI di AWS

AWS Panduan Preskriptif



AWS Panduan Preskriptif: Pola dan alur kerja Agentic AI di AWS

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

Merek dagang dan tampilan dagang Amazon tidak boleh digunakan sehubungan dengan produk atau layanan apa pun yang bukan milik Amazon, dengan cara apa pun yang dapat menyebabkan kebingungan di antara pelanggan, atau dengan cara apa pun yang merendahkan atau mendiskreditkan Amazon. Semua merek dagang lain yang tidak dimiliki oleh Amazon merupakan hak milik masing-masing pemiliknya, yang mungkin atau mungkin tidak terafiliasi, terkait dengan, atau disponsori oleh Amazon.

Table of Contents

Pengantar	1
Audiens yang dituju	1
Tujuan	1
Tentang seri konten ini	2
Pola agen	3
Agen penalaran dasar	4
Arsitektur	4
Deskripsi	5
Kemampuan	6
Batasan	6
Kasus penggunaan umum	6
Panduan implementasi	7
Ringkasan	7
Arsitektur	7
Deskripsi	8
Kemampuan	9
Kasus penggunaan umum	9
Panduan implementasi	9
Ringkasan	10
Tool-based agen untuk memanggil fungsi	10
Arsitektur	10
Deskripsi	11
Kemampuan	12
Kasus penggunaan umum	12
Panduan implementasi	12
Ringkasan	13
Tool-based agen untuk server	13
Arsitektur	13
Deskripsi	14
Kemampuan	15
Kasus penggunaan umum	15
Panduan implementasi	15
Ringkasan	16
Computer-use agen	16

Arsitektur	16
Deskripsi	17
Kemampuan	18
Kasus penggunaan umum	18
Panduan implementasi	19
Ringkasan	19
Agen pengkodean	19
Arsitektur	19
Deskripsi	20
Kemampuan	21
Kasus penggunaan umum	21
Panduan implementasi	22
Ringkasan	22
Agen pidato dan suara	22
Arsitektur	22
Deskripsi	23
Kemampuan	24
Kasus penggunaan umum	24
Panduan implementasi	24
Ringkasan	25
Agen orkestrasi alur kerja	25
Arsitektur	25
Deskripsi	26
Kemampuan	27
Kasus penggunaan umum	27
Panduan implementasi	27
Ringkasan	28
Memory-augmented agen	28
Arsitektur	28
Deskripsi	29
Kemampuan	30
Kasus penggunaan umum	30
Menerapkan agen yang diperbesar memori	30
Menerapkan dorongan yang disuntikkan memori	31
Ringkasan	31
Agen simulasi dan test-bed	32

Arsitektur	32
Deskripsi	33
Kemampuan	34
Kasus penggunaan umum	34
Panduan implementasi	35
Ringkasan	35
Pengamat dan agen pemantauan	36
Arsitektur	36
Deskripsi	36
Kemampuan	37
Kasus penggunaan umum	37
Panduan implementasi	38
Ringkasan	38
Multi-agent kolaborasi	39
Deskripsi	40
Kemampuan	41
Kasus penggunaan umum	41
Panduan implementasi	41
Ringkasan	42
Kesimpulan	42
Takeaways	43
Alur kerja LLM	44
Ikhtisar LLM-augmented kognisi	44
Alur kerja untuk rantai cepat	45
Deskripsi	46
Kemampuan	46
Kasus penggunaan umum	47
Alur kerja untuk perutean	47
Kemampuan	48
Kasus penggunaan umum	48
Alur kerja untuk paralelisasi	48
Kemampuan	50
Kasus penggunaan umum	50
Alur kerja untuk orkestrasi	50
Kemampuan	51
Kasus penggunaan umum	51

Alur kerja untuk evaluator dan loop refleksi-refine	52
Kasus penggunaan umum	53
Kemampuan	53
Kesimpulan	53
Pola alur kerja agen	55
Dari sistem yang didorong oleh peristiwa hingga peningkatan kognisi	55
Event-driven arsitektur	55
Cognition-augmented alur kerja	56
Wawasan inti	58
Pola saga rantai yang cepat	58
Koreografi Saga	59
Pola rantai yang cepat	60
Koreografi agen	60
Takeaways	62
Merutekan pola pengiriman dinamis	62
Pengiriman dinamis	63
LLM-based perutean	64
Router agen	65
Takeaways	66
Paralelisasi dan pola scatter-gathering	66
Scatter-gather	67
LLM-based paralelisasi (kognisi sebar-kumpulkan)	69
Paralelisasi agen	69
Takeaways	70
Pola orkestrasi Saga	70
Orkestrasi acara	71
Role-based sistem agen (orkestrator)	72
Pengawas	72
Takeaways	74
Evaluator refleksi-memperbaiki pola loop	74
Lingkaran kontrol umpan balik	75
Lingkaran kontrol umpan balik (evaluator)	77
Penilai	77
Takeaways	78
Merancang alur kerja agen pada AWS	78
Kesimpulan	79

Riwayat dokumen 80

..... lxxxi

Pola dan alur kerja AI agen di AWS

Aaron Sempf dan Andrew Hooker, Amazon Web Services

Juli 2025 ([sejarah dokumen](#))

Organizations mengadopsi model bahasa besar (LLMs) dan agen perangkat lunak untuk memecahkan masalah multidomain dinamis menggunakan disiplin arsitektur baru yang disebut pola agen. Pola agen adalah cetak biru dasar dan konstruksi modular yang digunakan untuk merancang dan mengatur agen AI yang berorientasi pada tujuan di banyak konteks.

Audiens yang dituju

Panduan ini ditujukan untuk arsitek, pengembang, dan pemimpin produk yang ingin membangun aplikasi cerdas yang melampaui logika statis, logika simbolik, dan otomatisasi deterministik.

Tujuan

Panduan ini menyediakan kerangka desain dan pendekatan implementasi untuk sistem agen AI yang beroperasi secara mandiri sambil tetap dapat dikontrol dan selaras dengan tujuan Anda. Ini menghubungkan pola arsitektur berbasis peristiwa dengan berbagai alternatif agen, menunjukkan cara membangun sistem agen tingkat produksi menggunakan arsitektur cloud-native. Mata pelajaran berikut dibahas dalam panduan ini:

- Pola agen — Pola agen adalah templat desain yang dapat digunakan kembali yang menggambarkan struktur dan perilaku agen individu. Ini termasuk agen penalaran, agen tambahan pengambilan, agen pengkodean, antarmuka suara, orkestra alur kerja, dan sistem multi-agen kolaboratif. Setiap pola menggambarkan bagaimana agen memandang, bernalar, bertindak, dan belajar, dipetakan. Layanan AWS
- Alur kerja LLM - Alur kerja fokus pada bagaimana agen menggunakan LLMs untuk penalaran. Mereka mengeksplorasi strategi yang mendorong dan mekanisme perencanaan, dan menguraikan bagaimana LLMs digunakan tidak hanya untuk menghasilkan teks tetapi juga untuk mendorong perilaku yang terstruktur, dapat ditafsirkan, dan dapat diandalkan dalam lingkaran agen.
- Pola alur kerja agen — Pola alur kerja menggambarkan bagaimana beberapa agen, alat, dan lingkungan berinteraksi untuk membentuk sistem otonom. Ini termasuk pola untuk orkestrasi tugas, delegasi subagen, koordinasi berbasis peristiwa, observabilitas, dan kontrol. Aspek-aspek ini mempromosikan arsitektur AI yang dapat diskalakan, dapat dikomposisi, dan dapat diaudit.

Tentang seri konten ini

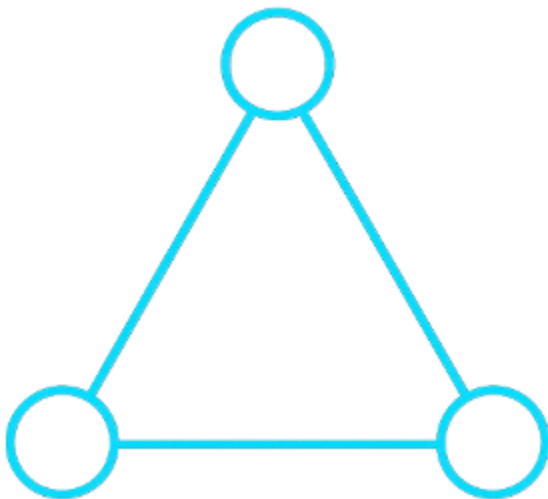
Panduan ini adalah bagian dari seri tentang AI agen di AWS. Untuk informasi lebih lanjut dan untuk melihat panduan lain dalam seri ini, lihat [Agentic AI](#) di situs web AWS Prescriptive Guidance.

Pola agen

Pola agen adalah blok bangunan yang dapat digunakan kembali dan dapat dikomposisi yang dapat disesuaikan dengan domain tertentu, kasus penggunaan, dan tingkat kompleksitas. Sistem agen berbeda, bagaimanapun, dari aplikasi tradisional. Inti dari semua desain agen AI adalah model konseptual yang berlabuh dalam tiga prinsip dasar berikut:

- Asinkron — Agen beroperasi di lingkungan yang kaya peristiwa yang digabungkan secara longgar
- Otonomi — Agen bertindak secara independen, tanpa kontrol manusia atau eksternal
- Agensi — Agen bertindak dengan tujuan, atas nama pengguna atau sistem, menuju tujuan tertentu

Segitiga dalam diagram berikut mewakili blok bangunan inti dari agen perangkat lunak: persepsi, alasan, dan tindakan. Hal ini memungkinkan sistem agen untuk mengamati, membuat keputusan, dan bertindak dalam lingkungannya.



Secara desain, pola agen menyediakan bahasa desain modular untuk membangun sistem AI, yang berarti mereka dapat diakses, operasional, dapat diperluas, dan siap produksi. Merancang sistem ini membutuhkan perhatian yang cermat terhadap tiga dimensi yang saling terkait berikut, yang selanjutnya dibahas dalam panduan ini.

Dalam bagian ini

- [Agen penalaran dasar](#)
- [Tool-based agen untuk memanggil fungsi](#)
- [Tool-based agen untuk server](#)

- [Computer-use agen](#)
- [Agen pengkodean](#)
- [Agen pidato dan suara](#)
- [Agen orkestrasi alur kerja](#)
- [Memory-augmented agen](#)
- [Agen simulasi dan test-bed](#)
- [Pengamat dan agen pemantauan](#)
- [Multi-agent kolaborasi](#)

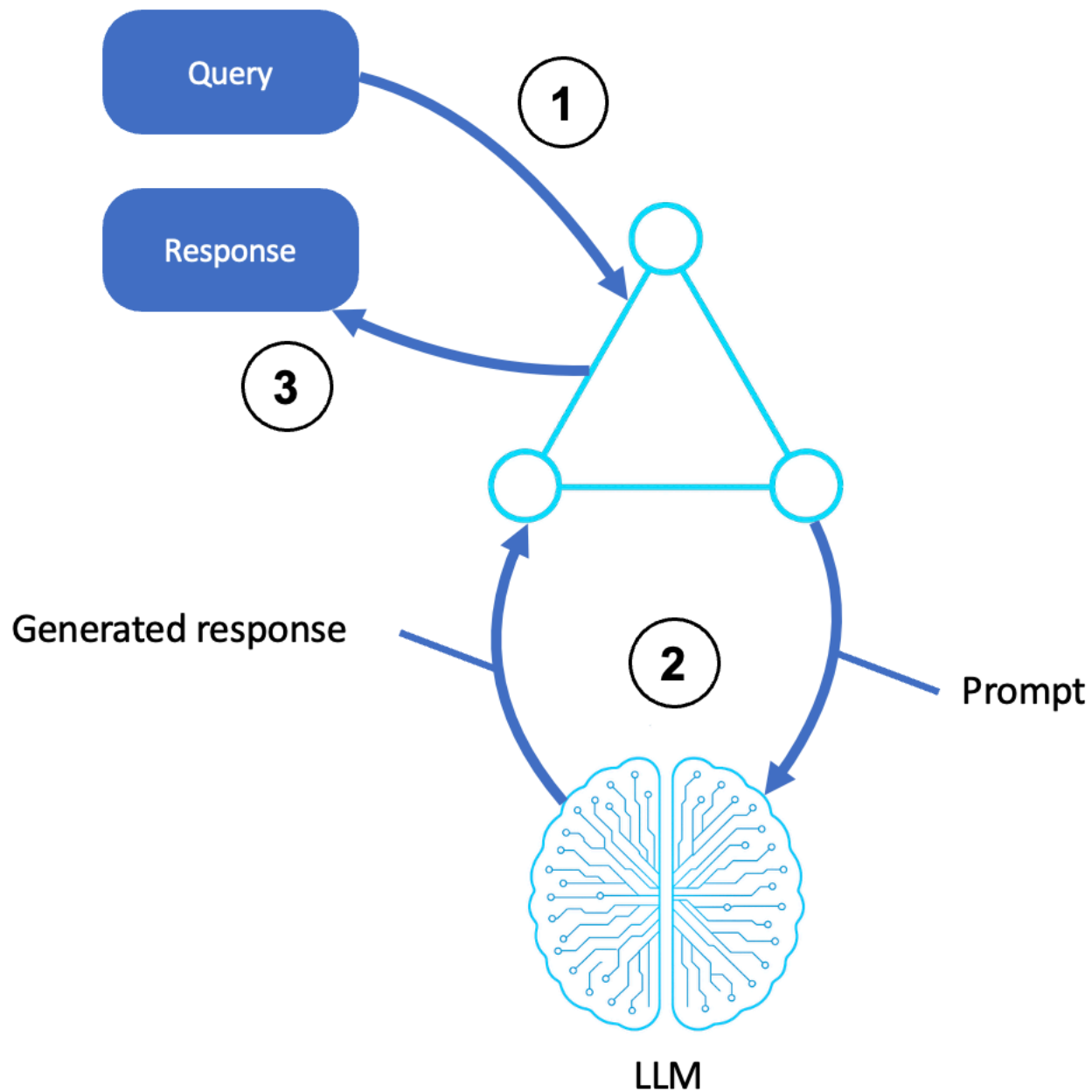
Agen penalaran dasar

Agen penalaran dasar adalah bentuk paling sederhana dari AI agen yang melakukan inferensi logis atau pengambilan keputusan dalam menanggapi kueri. Ini menerima masukan dari pengguna atau sistem dan memproses kueri dan menghasilkan tanggapan menggunakan prompt terstruktur.

Pola ini berguna untuk tugas-tugas yang membutuhkan penalaran, klasifikasi, atau ringkasan satu langkah berdasarkan konteks tertentu. Itu tidak menggunakan memori, alat, atau manajemen status, yang membuatnya stateless, ringan, dan sangat dapat dikomposisi di seluruh alur kerja besar.

Arsitektur

Aliran agen penalaran dasar ditunjukkan pada diagram berikut:



Deskripsi

1. Menerima masukan

- Pengguna, sistem, atau agen hulu mengirimkan kueri atau instruksi.
- Input diserahkan ke shell agen atau lapisan orkestrasi.
- Langkah ini mencakup setiap preprocessing, prompt templating, dan identifikasi tujuan.

2. Memanggil LLM

- Agen mengubah kueri menjadi prompt terstruktur dan mengirimkannya ke LLM (misalnya, melalui Amazon Bedrock).
- LLM menghasilkan respons berdasarkan prompt menggunakan pengetahuan dan konteks yang telah dilatih sebelumnya.
- Output yang dihasilkan dapat mencakup langkah-langkah penalaran (rantai pemikiran), jawaban akhir, atau opsi peringkat.

3. Mengembalikan respon

- Output yang dihasilkan diteruskan ke antarmuka agen.
- Ini mungkin termasuk pemformatan, postprocessing, atau respons API.

Kemampuan

- Mendukung bahasa alami atau masukan terstruktur
- Menggunakan rekayasa cepat untuk memandu perilaku
- Tanpa kewarganegaraan dan terukur
- Dapat disematkan ke UI, CLI, API, dan pipeline

Batasan

- Tidak ada ingatan atau kesadaran historis
- Tidak ada interaksi dengan alat eksternal atau sumber data
- Terbatas pada apa yang diketahui LLM pada saat inferensi

Kasus penggunaan umum

- Pertanyaan dan jawaban percakapan
- Penjelasan dan ringkasan kebijakan
- Panduan untuk membuat keputusan
- Alur chatbot yang ringan dan otomatis
- Klasifikasi, pelabelan, dan penilaian

Panduan implementasi

Anda dapat menggunakan alat dan layanan berikut untuk membuat agen penalaran dasar:

- Amazon Bedrock untuk doa LLM (Antropik, AI21, Meta)
- Amazon API Gateway atau AWS Lambda untuk mengeksposnya sebagai layanan mikro tanpa kewarganegaraan
- Template prompt disimpan di Parameter Store AWS Secrets Manager,, atau sebagai kode

Ringkasan

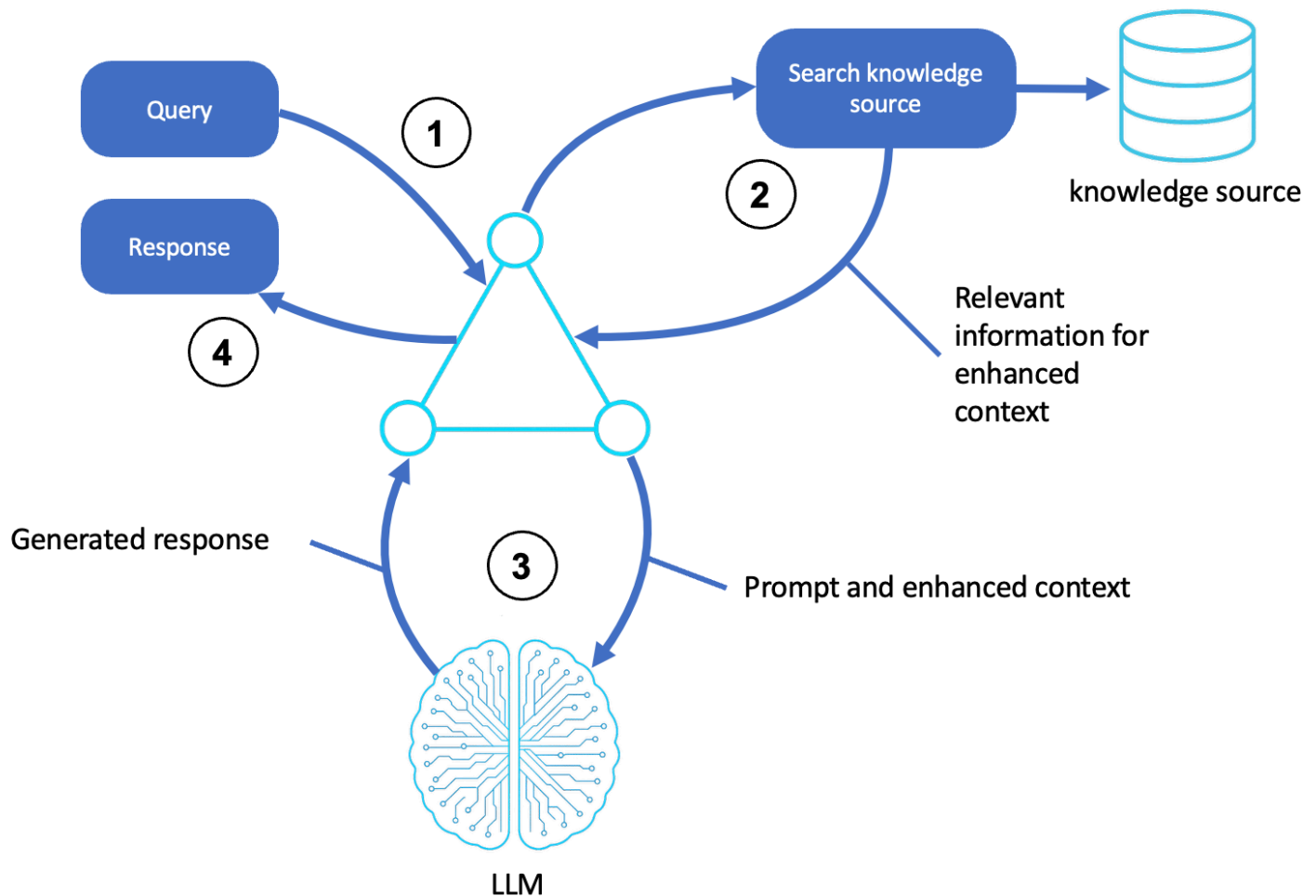
Agan penalaran dasar adalah dasar karena strukturnya yang sederhana. Ini memiliki kemampuan inti yang mengubah tujuan menjadi jalur penalaran yang mengarah pada output cerdas. Pola ini sering menjadi titik awal untuk pola lanjutan, seperti agen berbasis alat dan agen yang menggunakan retrieval-augmented generation (RAG). Ini juga merupakan komponen yang andal dan modular dari alur kerja besar.

Agan RAG

Retrieval-augmented Generasi (RAG) adalah teknik yang menggabungkan pengambilan informasi dengan pembuatan teks untuk menciptakan respons yang akurat dan kontekstual. RAG memungkinkan agen untuk mengambil informasi eksternal yang relevan sebelum melibatkan LLM. Ini memperluas memori efektif agen dan akurasi penalaran dengan membumikan keputusannya dalam informasi terkini, faktual, atau spesifik domain. Berbeda dengan LLM stateless yang hanya mengandalkan bobot yang telah dilatih sebelumnya, RAG memiliki lapisan pencarian pengetahuan eksternal yang secara dinamis meningkatkan permintaan dengan konteks.

Arsitektur

Logika pola RAG diilustrasikan dalam diagram berikut:



Deskripsi

1. Menerima kueri

- Pengguna atau sistem hulu mengirimkan kueri atau tujuan ke agen.
- Shell agen menerima permintaan dan memformatnya sebagai prompt untuk penalaran.

2. Mencari sumber eksternal

- Agen mengidentifikasi konsep dan maksud dari kueri.
- Ini menanyakan sumber pengetahuan, seperti penyimpanan vektor, database, atau indeks dokumen menggunakan pencarian semantik atau pencocokan kata kunci.
- Bagian, dokumen, atau entitas yang paling relevan diambil untuk digunakan pada langkah berikutnya.

3. Menghasilkan respons kontekstual

- Agen menambah prompt dengan informasi yang diambil, membentuk input yang ditingkatkan konteks untuk LLM.
- LLM memproses setiap input menggunakan penalaran generatif (misalnya, rantai pemikiran atau refleksi) untuk menghasilkan respons yang akurat.

4. Mengembalikan output akhir

- Agen menyiapkan output dengan membungkusnya dalam header komunikasi atau format yang diperlukan dan kemudian mengembalikannya ke pengguna atau sistem panggilan.
- (Opsional) Dokumen yang diambil dan keluaran LLM dapat dicatat, dinilai, dan disimpan dalam memori untuk kueri masa depan.

Kemampuan

- Fact-grounded output bahkan dalam domain ekor panjang atau khusus perusahaan
- Ekstensi memori tanpa menyempurnakan model
- Konteks dinamis berdasarkan setiap kueri dan status pengguna
- Sepenuhnya kompatibel dengan database vektor, indeks semantik, dan penyaringan metadata

Kasus penggunaan umum

- Asisten pengetahuan perusahaan
- Bot kepatuhan peraturan
- Copilot dukungan pelanggan
- Search-enhanced chatbots
- Agen dokumentasi pengembang

Panduan implementasi

Gunakan alat dan layanan berikut untuk membuat agen yang menggunakan RAG:

- Amazon Bedrock untuk pemanggilan LLM
- Amazon Kendra, OpenSearch, atau Amazon Aurora untuk dokumentasi atau pencarian data terstruktur
- Amazon Simple Storage Service (Amazon S3) untuk penyimpanan dokumen

- AWS Lambda untuk mengatur pencarian, prompt, dan inferensi LLM
- Knowledge-based integrasi dengan agen (dengan menggunakan plugin memori, retriever semantik, atau Amazon Bedrock)

Ringkasan

Agan RAG menghubungkan penalaran model statis dengan kecerdasan dunia nyata yang dinamis. Ini melengkapi agen dengan kemampuan untuk mencari apa yang tidak mereka ketahui, mensintesis jawaban dari pengetahuan yang diambil, dan menghasilkan respons yang dapat diaudit dengan kepercayaan tinggi.

Pola RAG adalah fondasi untuk membangun agen cerdas yang meningkatkan akses pengetahuan tanpa pelatihan ulang. Ini sering merupakan pendahulu pola orkestrasi yang lebih kompleks yang melibatkan penggunaan alat, perencanaan, dan memori jangka panjang.

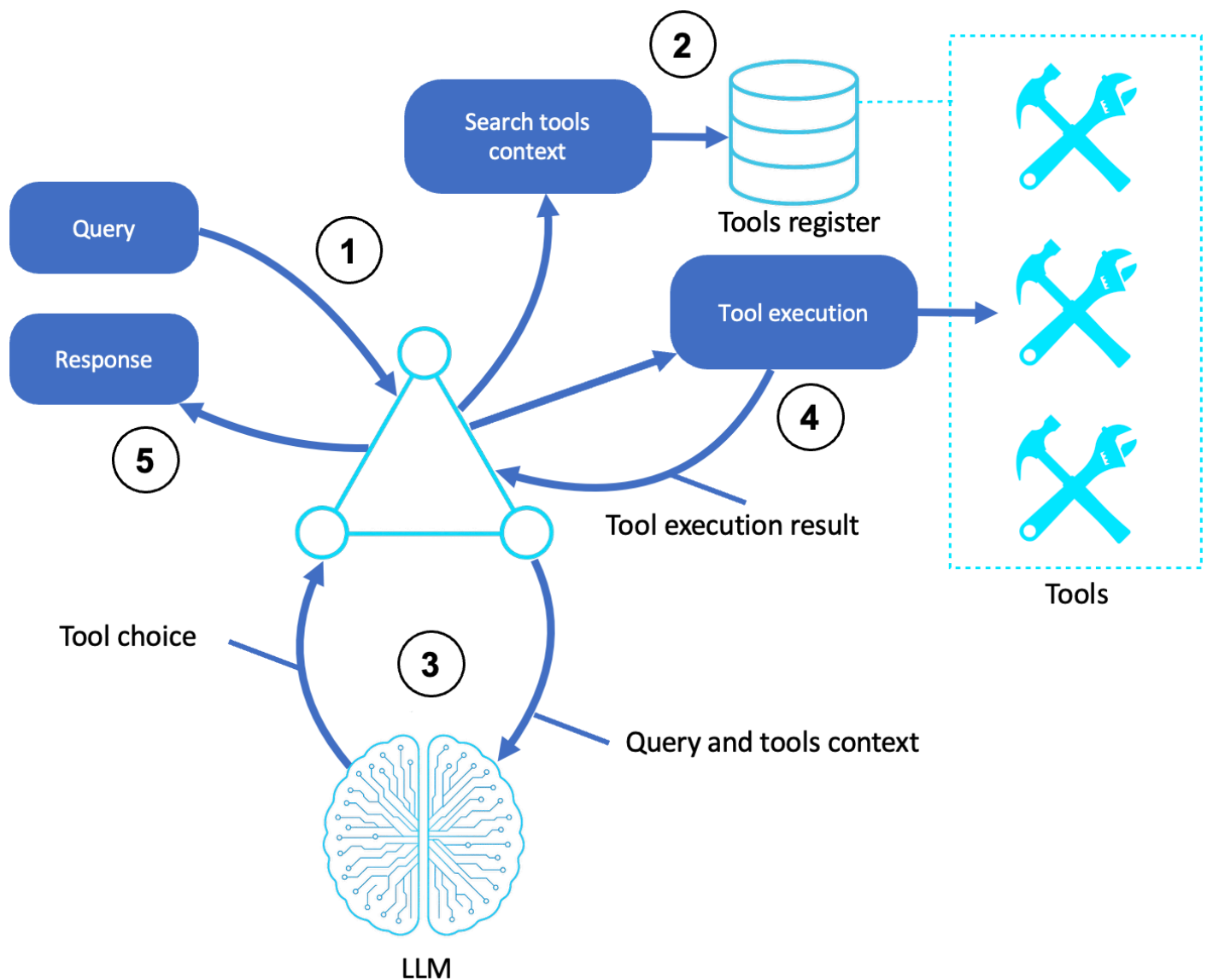
Tool-based agen untuk memanggil fungsi

Tool-based agen memperluas kemampuan agen penalaran dengan menggunakan fungsi eksternal atau API untuk menyelesaikan tugas yang melampaui penalaran bahasa saja. Pola ini menggunakan LLM untuk memutuskan alat mana yang akan digunakan dan kemudian menghasilkan argumen panggilan dan menggabungkan output alat ke dalam loop penalarannya.

Pola ini memungkinkan agen untuk bertindak daripada hanya memberikan tanggapan. Antarmuka alat mewakili kemampuan yang dapat dipanggil, mulai dari perhitungan aritmatika dan pencarian database hingga API eksternal dan layanan cloud.

Arsitektur

Agan berbasis alat untuk memanggil fungsi ditunjukkan dalam diagram berikut:



Deskripsi

1. Menerima kueri

- Agen menerima kueri atau tugas bahasa alami dari pengguna atau sistem panggilan.

2. Telusuri alat

- Agen menggunakan metadata internal atau registri alat untuk mencari alat, skema, dan kemampuan yang relevan yang tersedia.

3. Memilih dan memanggil alat

- LLM menerima kueri dan metadata alat (misalnya, nama fungsi, jenis input, dan deskripsi) dalam prompt nya.

- Ia memilih alat yang paling relevan, membangun argumen masukan, dan mengembalikan panggilan fungsi terstruktur.
4. Menjalankan alat yang dipilih
 - Shell agen atau tool runner mengeksekusi fungsi yang dipilih dan mengembalikan hasilnya (misalnya, output API, nilai database, atau komputasi).
 5. Mengembalikan respon
 - LLM meneruskan hasil ke agen, baik secara langsung atau sebagai bagian dari prompt yang diperbarui. Kemudian mengembalikan hasil bahasa alami.

Kemampuan

- Pemilihan alat dinamis berdasarkan konteks tugas
- Schema-based meminta (OpenAPI, skema AWS JSON, antarmuka fungsi)
- Interpretasi hasil dan rantai output ke dalam penalaran
- Operasi tanpa kewarganegaraan atau sadar sesi

Kasus penggunaan umum

- Asisten virtual dengan akses data eksternal
- Kalkulator dan estimator keuangan
- API-based pekerja pengetahuan
- LLM yang memanggil, SageMaker titik akhir AWS Lambda Amazon, dan layanan SaaS

Panduan implementasi

Gunakan yang berikut ini untuk membuat agen berbasis alat untuk memanggil fungsi:

- Amazon Bedrock dengan dukungan pemanggilan fungsi (Anthropic Claude)
- AWS Lambda sebagai backend eksekusi alat
- Amazon API Gateway atau AWS Step Functions untuk orkestrasi alat
- Amazon DynamoDB atau Amazon Relational Database Service (Amazon RDS) untuk metadata alat sadar konteks

- EventBridge Saluran pipa Amazon atau peta AWS Step Functions yang menyatakan untuk merutekan output

Ringkasan

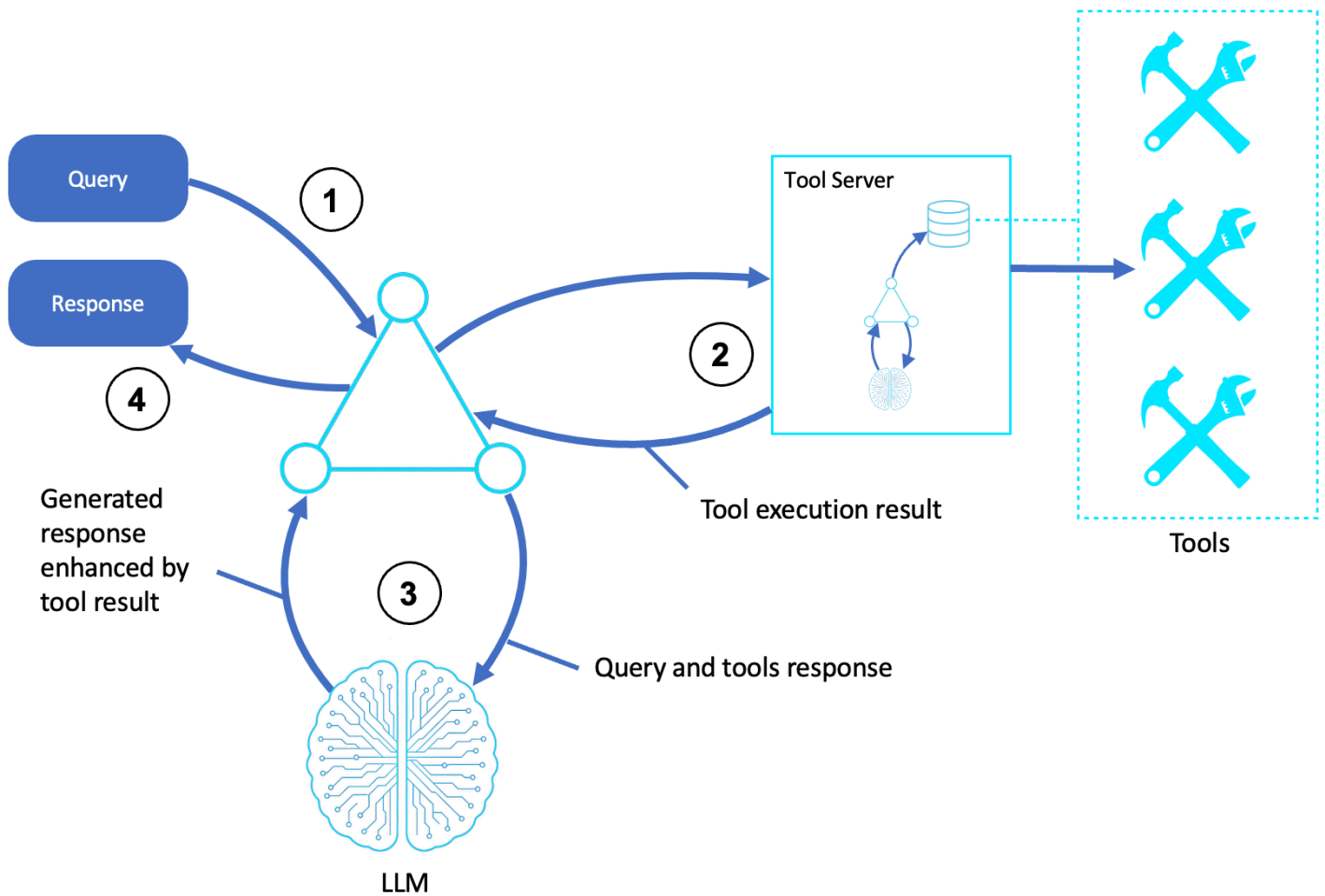
Tool-based agen pemanggil fungsi mewakili pergeseran dari pemahaman bahasa ke melakukan tindakan. Agen-agen ini menggunakan alat yang dinamis dan sadar konteks sambil mempertahankan penalaran LLM, mengubah asisten pasif menjadi sistem yang menyelesaikan tugas, mengakses layanan, dan mengintegrasikan operasi bisnis. Pola ini merupakan komponen penting dari AI agen dalam pengaturan perusahaan, terutama bila dikombinasikan dengan skema deklaratif, kerangka kerja otorisasi, dan sistem multi-agen.

Tool-based agen untuk server

Tool-based agen untuk server meningkatkan agen pemanggil fungsi dengan mendelegasikan eksekusi alat ke server eksternal yang memiliki lingkungan runtime khusus untuk alat, skrip, dan agen komposit. Tidak seperti panggilan fungsi inline yang dipilih dan dipanggil oleh loop agen, agen berbasis server mengalihdayakan logika dan pipa eksekusi ke agen atau sistem lain. Ini memberikan kemampuan canggih seperti rantai multitool, eksekusi terisolasi, dan penalaran khusus. Server alat ideal untuk tindakan yang kompleks, stateful, atau intensif sumber daya di mana alat itu sendiri mungkin melibatkan model AI, aturan bisnis, atau lingkungan yang terpisah.

Arsitektur

Berikut ini adalah pola untuk agen berbasis alat untuk server:



Deskripsi

1. Menerima kueri

- Pengguna atau sistem mengirimkan permintaan ke shell agen.
- Agen menafsirkan kueri dan bersiap untuk mengirimkannya ke server alat.

2. Menjalankan proses server alat

- Agen mengirimkan tugas, bersama dengan parameter terstruktur, ke server alat.
- Server alat kemudian dapat:
 - Jalankan skrip atau logika dalam sistem komputasi khusus (misalnya, wadah AWS Lambda, atau Amazon) SageMaker
 - Gunakan subagennya sendiri dengan penalaran LLM untuk memilih dan menjalankan alat
 - Mengelola dependensi, percobaan ulang, atau alur eksekusi multistep
 - Hasil keluaran ke agen utama saat tugas selesai

3. Menggunakan penalaran LLM dengan keluaran alat

- Agen memanggil LLM, meneruskan kueri asli dan hasil server alat sebagai bagian dari prompt.
- LLM mensintesis respons yang menggabungkan informasi yang baru diperoleh.

4. Mengembalikan respon

- Agen mengembalikan respons bahasa alami atau terstruktur ke pengguna atau sistem panggilan.
- (Opsional) Hasil dapat disimpan dalam memori atau log audit.

Kemampuan

- Alat dipanggil di luar loop eksekusi agen utama
- Eksekusi alat mungkin melibatkan panggilan LLM, rantai logika, atau subagen
- Agen bertindak sebagai pengontrol atau operator, bukan hanya pembungkus alat
- Mengaktifkan komposabilitas, skalabilitas, dan isolasi logika

Kasus penggunaan umum

- Mengatur rantai model (misalnya, dengan menggabungkan LLM, visi, dan kode)
- AI-driven jaringan pipa otomatisasi
- DevOps asisten agen dengan pelari skrip
- Kompleks perhitungan keuangan, simulasi, atau agen optimasi
- Alat multimodal (misalnya, dengan menggabungkan audio, dokumentasi, dan tindakan)

Panduan implementasi

Anda dapat membangun pola ini menggunakan yang berikut Layanan AWS:

- Amazon Bedrock (host agen dan inferensi LLM)
- AWS Lambda, Amazon ECS, AWS Fargate, atau SageMaker titik akhir Amazon sebagai runtime server alat
- Amazon API Gateway atau AWS App Runner untuk mengekspos API server alat
- Amazon EventBridge untuk pesan agen-ke-alat yang dipisahkan
- AWS Step Functions atau AWS AppFabric untuk menyusun logika multi-agen di server alat

Ringkasan

Tool-based agen yang menggunakan server sangat modular dan dapat diskalakan. Mereka memisahkan logika keputusan dari eksekusi, yang memungkinkan agen utama tetap ringan saat membongkar tindakan kompleks atau sensitif ke sistem lain. Ini penting untuk AI agen tingkat perusahaan, terutama di lingkungan yang membutuhkan tata kelola, observabilitas, isolasi, komposisi dinamis, atau kombinasi apa pun darinya.

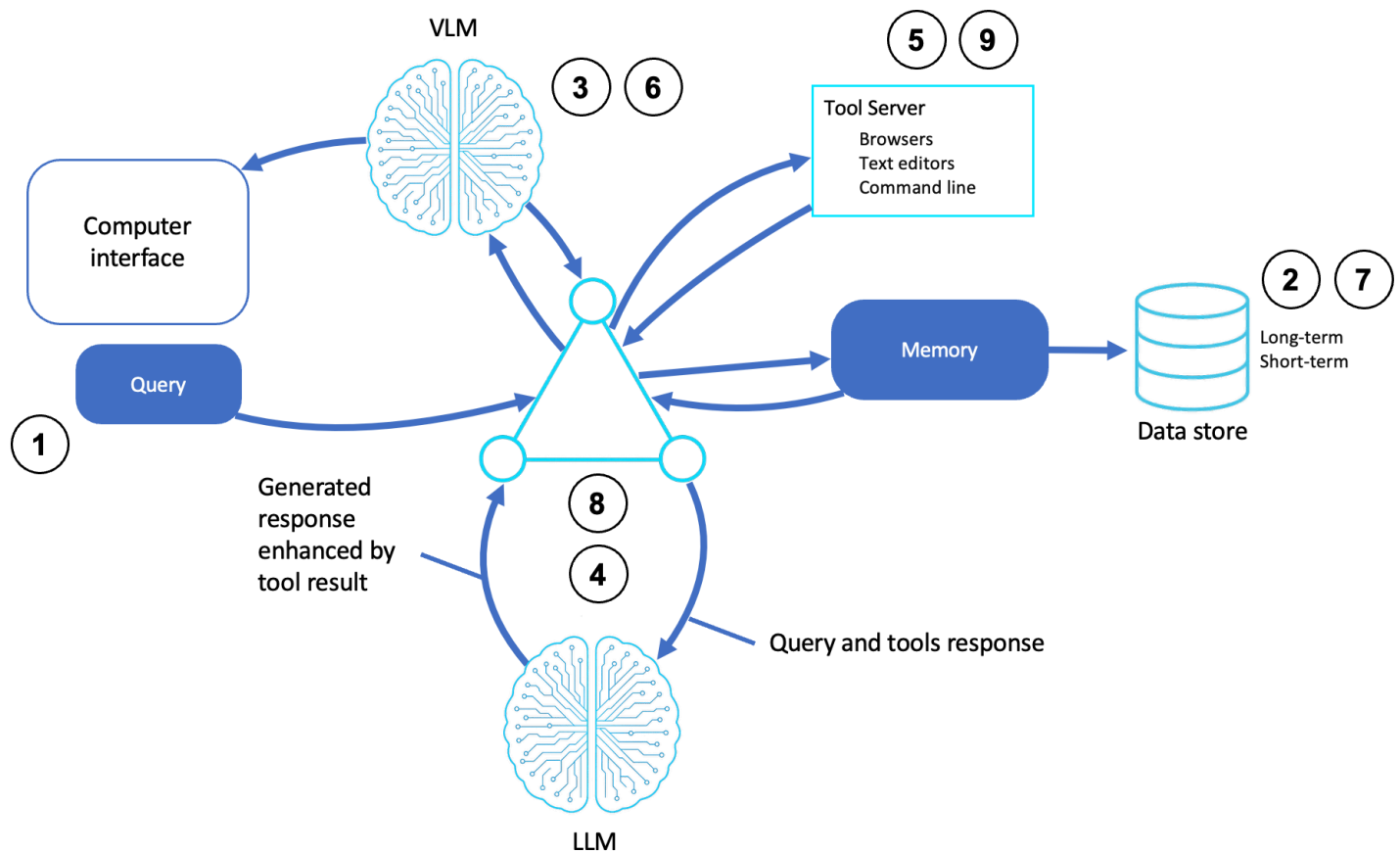
Computer-use agen

Computer-use agen dapat mensimulasikan atau mengontrol lingkungan digital seperti browser, terminal, sistem file, dan aplikasi. Agen ini menafsirkan maksud pengguna, berinteraksi dengan antarmuka visual dan teks, dan melakukan tindakan yang diarahkan pada tujuan dengan menggabungkan penalaran LLM, model bahasa visual (VLM), dan server alat yang menjalankan perintah atau mensimulasikan peristiwa input.

Pola ini penting untuk otomatisasi AI praktis, di mana agen berfungsi tidak hanya sebagai asisten tetapi juga sebagai proxy yang melakukan tindakan seperti manusia, seringkali dengan menggunakan alat dan lingkungan yang sama.

Arsitektur

Pola agen penggunaan komputer ditunjukkan pada diagram berikut:



Deskripsi

1. Menerima kueri

- Tugas atau permintaan disediakan melalui UI, API, atau antarmuka bahasa alami.

2. Mengakses memori

- Agen mengambil memori jangka pendek dan jangka panjang untuk mengingat perintah, tujuan, dan status sistem masa lalu.

3. Menganalisis konteks visual

- VLM mengamati layar komputer, status sistem, atau elemen UI untuk memahami konteks tertentu dan mengidentifikasi item yang dapat ditindaklanjuti.

4. Alasan melalui LLM

- LLM menggabungkan kueri, status memori, alat, dan respons server untuk menentukan tindakan selanjutnya.

5. Berinteraksi dengan server alat

- Agen memanggil alat yang di-host di server, yang mungkin termasuk yang berikut:

- Browser (misalnya, Chrome tanpa kepala) dan lingkungan shell
 - Editor teks dan kode
 - Antarmuka skrip kustom
6. Memperbarui input visual
- Jika UI sistem berubah atau pengamatan lebih lanjut diperlukan, VLM dapat menganalisis ulang status layar atau buffer teks.
7. Memperbarui memori
- Wawasan baru, status sistem, atau umpan balik pengguna ditulis ke memori jangka pendek dan jangka panjang.
8. Merumuskan keputusan akhir dan penjelasan
- LLM mensintesis hasil atau merekomendasikan tindakan berdasarkan kueri dan output alat.
9. Mengembalikan respon
- Agen mengembalikan hasil ke antarmuka (misalnya, tugas yang diselesaikan, konfirmasi, atau konten yang dihasilkan).

Kemampuan

- Penalaran multimodal dengan input visual dan tekstual
- Kontrol atas aplikasi melalui simulasi atau input API-driven
- Manajemen memori untuk keadaan persisten
- Otonomi dalam eksekusi urutan (aliran multistep)

Kasus penggunaan umum

- Pengembang AI yang menulis dan menjalankan kode di IDE
- Computer-use agen untuk alur kerja digital berulang
- Pengguna simulasi untuk pengujian perangkat lunak dan jaminan kualitas
- Agen aksesibilitas untuk menavigasi UI melalui suara atau instruksi tingkat tinggi
- Otomatisasi proses robot cerdas (RPA) yang ditingkatkan dengan penalaran

Panduan implementasi

- Anda dapat membangun pola ini menggunakan yang berikut Layanan AWS:
- Amazon Bedrock untuk LLM-based perencanaan dan penalaran
- Amazon Elastic Compute Cloud (Amazon EC2) AWS Lambda,, atau notebook SageMaker Amazon untuk menjalankan server alat dengan lingkungan UI simulasi
- Amazon Simple Storage Service (Amazon S3) atau Amazon DynamoDB untuk persistensi memori
- Amazon Rekognition (atau model kustom) untuk analisis gambar UI dalam skenario hybrid
- CloudWatch Log Amazon atau AWS X-Ray untuk jalur observabilitas dan audit

Ringkasan

Computer-use Agen bertindak sebagai operator digital otonom, menjembatani kesenjangan antara interaksi manusia-komputer dan tindakan. AI-driven Dengan menggabungkan memori, orkestrasi alat, dan VLM, agen ini dapat berinteraksi secara adaptif dengan sistem yang dirancang untuk manusia, menjalankan tindakan, memperbarui file, menavigasi menu, dan menghasilkan respons.

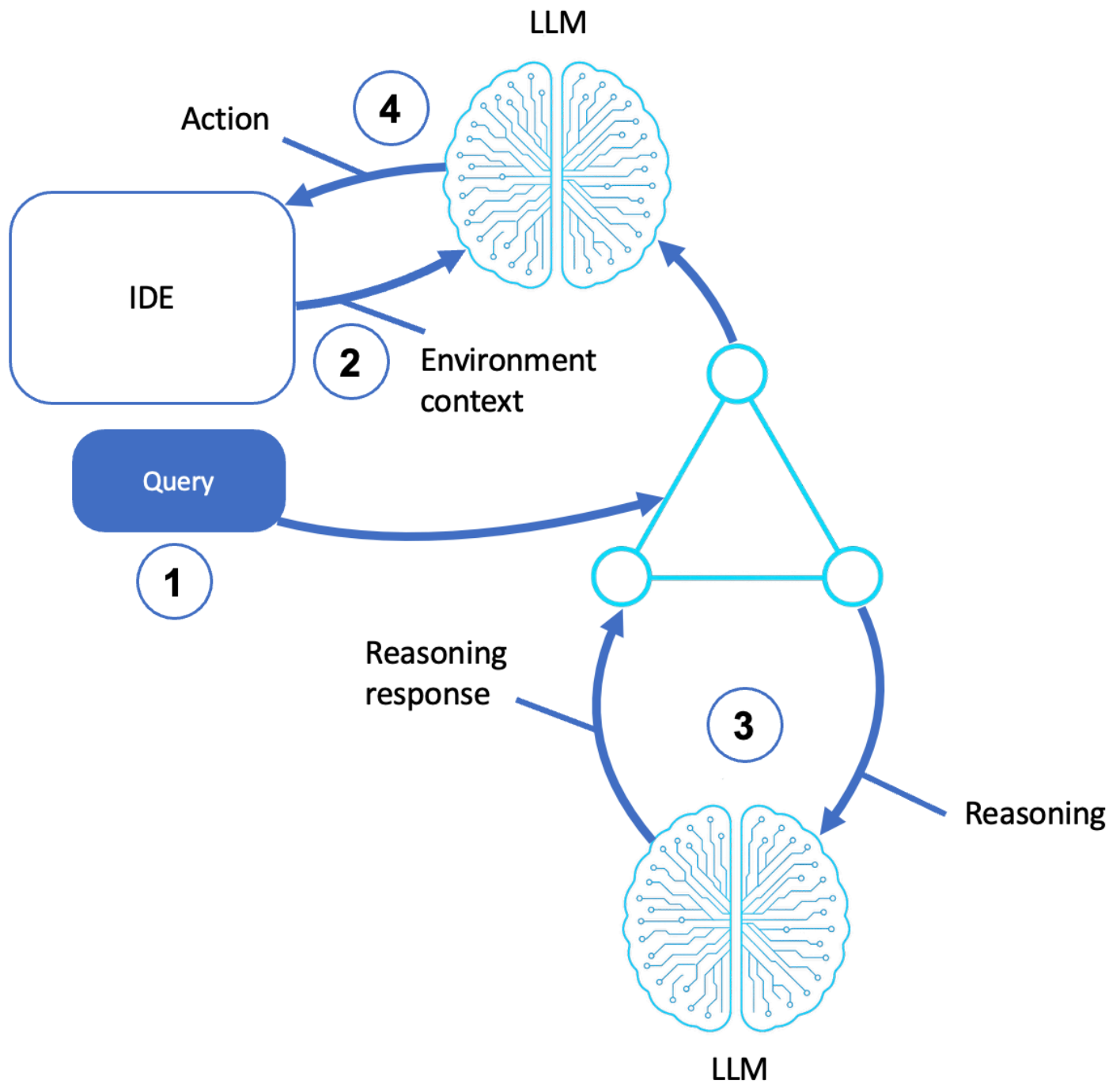
Agen pengkodean

Agen pengkodean dapat beralasan tentang tugas pemrograman, menghasilkan atau memodifikasi kode, dan berinteraksi dengan lingkungan pengembang, seperti IDE dan CLI. Agen-agen ini menggabungkan pemahaman bahasa alami dengan penalaran terstruktur untuk membantu, menambah, dan mengotomatiskan pengembangan perangkat lunak, mulai dari pembuatan fungsi hingga perbaikan bug dan penulisan pengujian.

Tidak seperti alat pelengkapan otomatis, agen pengkodean secara aktif menafsirkan tujuan pengguna, menanyakan lingkungan pengembangan untuk konteks (misalnya, membuka file dan melacak kesalahan), mengidentifikasi persyaratan, dan kemudian mengusulkan dan melakukan tindakan.

Arsitektur

Pola agen pengkodean ditunjukkan pada diagram berikut:



Deskripsi

1. Menerima kueri

- Pengguna menyediakan instruksi bahasa alami melalui palet perintah, jendela obrolan, atau CLI (misalnya, "Tambahkan logging ke fungsi ini" atau "Refactor untuk keterbacaan").

2. Ekstrak konteks lingkungan

- Agen mengumpulkan konteks dari IDE, termasuk file aktif, posisi kursor, cuplikan kode, dan tabel simbol.
- Ini mengeluarkan pesan kesalahan, hasil pengujian, dan output dari agen lain.

3. Alasan LLM

- Agen mengirimkan prompt, termasuk kueri dan konteks lingkungan, ke LLM.
 - LLM melakukan pass penalaran untuk menentukan hal-hal berikut:
 - Apa yang perlu diubah
 - Cara menghasilkan solusi
 - Setiap langkah refactoring, penulisan ulang, atau pengkodean

4. Mengeksekusi tindakan

- LLM mengembalikan output ke agen dan mengimpornya ke IDE atau lingkungan runtime.
- Ini mungkin termasuk memasukkan atau memodifikasi kode, membuat komentar atau dokumentasi, dan memicu tugas build, pengujian, dan linting hilir.

Kemampuan

- High-contextual kesadaran (misalnya, status IDE, kursor, dan pohon sintaks)
- Penalaran berulang dari tujuan dan umpan balik
- Perencanaan kode opsional dan pemisahan tindakan (misalnya, alasan pertama dan kemudian bertindak)
- Bekerja dalam alur kerja pengembang sinkron atau asinkron

Kasus penggunaan umum

- Pembuatan kode dari deskripsi tugas
- Pemfaktoran ulang dan pengoptimalan kode
- Test-case generasi dan validasi
- Penjelasan kesalahan dan debugging
- Asisten dokumentasi
- Kopilot pemrograman berpasangan

Panduan implementasi

- Anda dapat membangun pola ini menggunakan alat-alat berikut dan Layanan AWS:
- Amazon Bedrock untuk LLM-driven generasi dan penalaran
- Pengembang Amazon Q untuk saran dan penyelesaian pengkodean
- AWS Lambda atau Amazon Elastic Container Service (Amazon ECS) untuk menjalankan dan menguji lingkungan sandbox
- AWS Cloud9, ekstensi VS Code, atau integrasi IDE khusus untuk meng-host dan mengevaluasi konteks
- Amazon Simple Storage Service (Amazon S3) untuk menyimpan prompt perantara, respons, dan riwayat revisi

Ringkasan

Coding agent adalah alat AI-powered pengembangan baru yang mampu menafsirkan bahasa alami, menganalisis konteks, menghasilkan perubahan kode multistep, dan mengintegrasikan dengan siklus hidup pengembangan perangkat lunak.

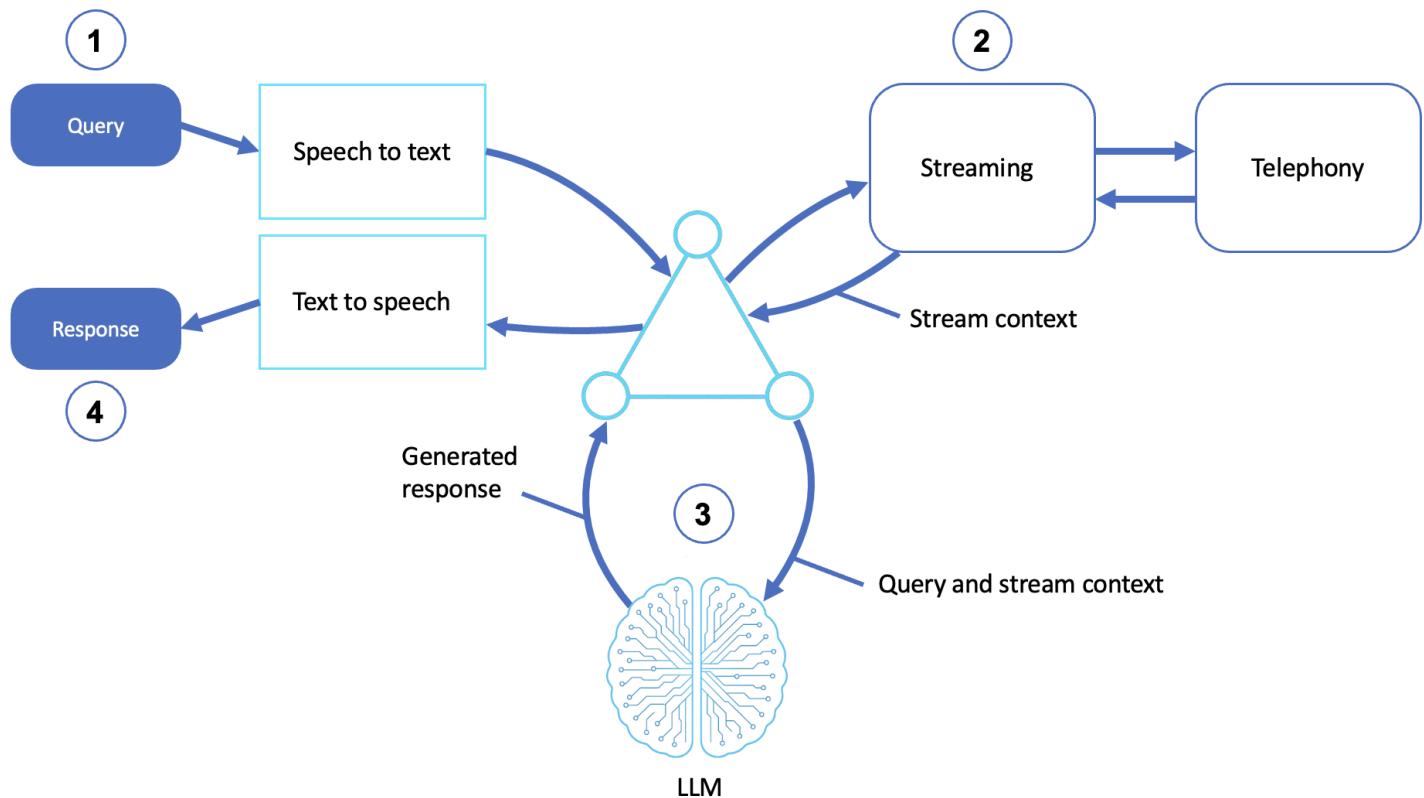
Agen pidato dan suara

Agen pidato dan suara berinteraksi dengan pengguna melalui dialog lisan. Agen-agen ini mengintegrasikan pengenalan suara, pemahaman bahasa alami, dan sintesis ucapan untuk memungkinkan AI percakapan di seluruh platform telepon, seluler, web, dan tertanam.

Agen suara sangat efektif dalam lingkungan hands-free, real-time, atau berbasis aksesibilitas. Dengan menggabungkan antarmuka streaming dengan LLM-powered penalaran, mereka memfasilitasi interaksi yang kaya dan dinamis yang terasa alami bagi pengguna.

Arsitektur

Agen pidato dan suara ditunjukkan dalam diagram berikut:



Deskripsi

1. Menerima kueri suara

- Pengguna menyuarakan permintaan ke telepon, mikrofon, atau sistem tertanam.
- Modul speech-to-text (STT) mengubah audio menjadi teks.

2. Mengintegrasikan konteks streaming dan telepon

- Agen menggunakan antarmuka streaming untuk mengelola audio I/O secara real time.
- Jika digunakan di pusat kontak atau konteks telekomunikasi, integrasi telepon menangani perutean sesi, input multi-frekuensi nada ganda (DTMF), dan transportasi media.

Catatan: DTMF mengacu pada nada yang dihasilkan saat Anda menekan tombol pada keypad telepon. Dalam konteks integrasi konteks streaming dan telepon dalam agen suara, DTMF digunakan sebagai mekanisme input sinyal selama panggilan telepon, terutama dalam sistem respons suara interaktif (IVR). Input DTMF memungkinkan agen untuk:

- Kenali pilihan menu (misalnya, “Tekan 1 untuk penagihan. Tekan 2 untuk dukungan.”)
- Kumpulkan input numerik (misalnya, nomor akun, PIN, dan nomor konfirmasi)

- Memicu alur kerja atau transisi status dalam alur panggilan
- Kembalikan dari ucapan ke nada sentuh bila perlu

1. Alasan melalui konteks aliran LLM

- Kueri dikirim ke agen, yang meneruskannya, bersama dengan metadata sesi apa pun (misalnya, ID pemanggil, konteks sebelumnya), ke LLM.
- LLM menghasilkan respons, mungkin menggunakan strategi rantai pemikiran atau memori multiturn jika interaksi sedang berlangsung.

2. Mengembalikan respons suara

- Agen mengubah responsnya terhadap ucapan menggunakan text-to-speech (TTS).
- Ini mengembalikan audio ke pengguna melalui saluran suara.

Kemampuan

- Real-time pemahaman pidato dan generasi
- Multilingual I/O dengan dukungan STT dan TTS
- Integrasi dengan API telepon atau streaming
- Kesadaran sesi dan serah terima memori di antara belokan

Kasus penggunaan umum

- Sistem IVR percakapan
- Resepsionis virtual dan penjadwal janji temu
- Voice-driven agen helpdesk
- Asisten suara yang dapat dikenakan
- Antarmuka suara untuk rumah pintar dan alat aksesibilitas

Panduan implementasi

Anda dapat membangun pola ini menggunakan alat-alat berikut dan Layanan AWS:

- Amazon Lex V2 atau Amazon Transcribe untuk STT
- Amazon Polly untuk TTS

- Amazon Chime SDK, Pelanggan Amazon Connect, atau Amazon Interactive Video Service (Amazon IVS) untuk streaming dan telepon
- Amazon Bedrock untuk penalaran dengan Anthropic, AI21, atau model yayasan lainnya
- AWS Lambda untuk menghubungkan STT, LLM, TTS, dan konteks sesi

(Opsional) Peningkatan tambahan dapat mencakup yang berikut:

- Amazon Kendra atau OpenSearch untuk RAG yang sadar konteks
- Amazon DynamoDB untuk memori sesi
- CloudWatch Log Amazon dan AWS X-Ray untuk ketertelusuran

Ringkasan

Agen bicara dan suara adalah sistem cerdas yang berinteraksi melalui percakapan alami. Dengan mengintegrasikan antarmuka ucapan dengan penalaran LLM dan infrastruktur streaming waktu nyata, agen suara memungkinkan interaksi yang mulus, dapat diakses, dan terukur.

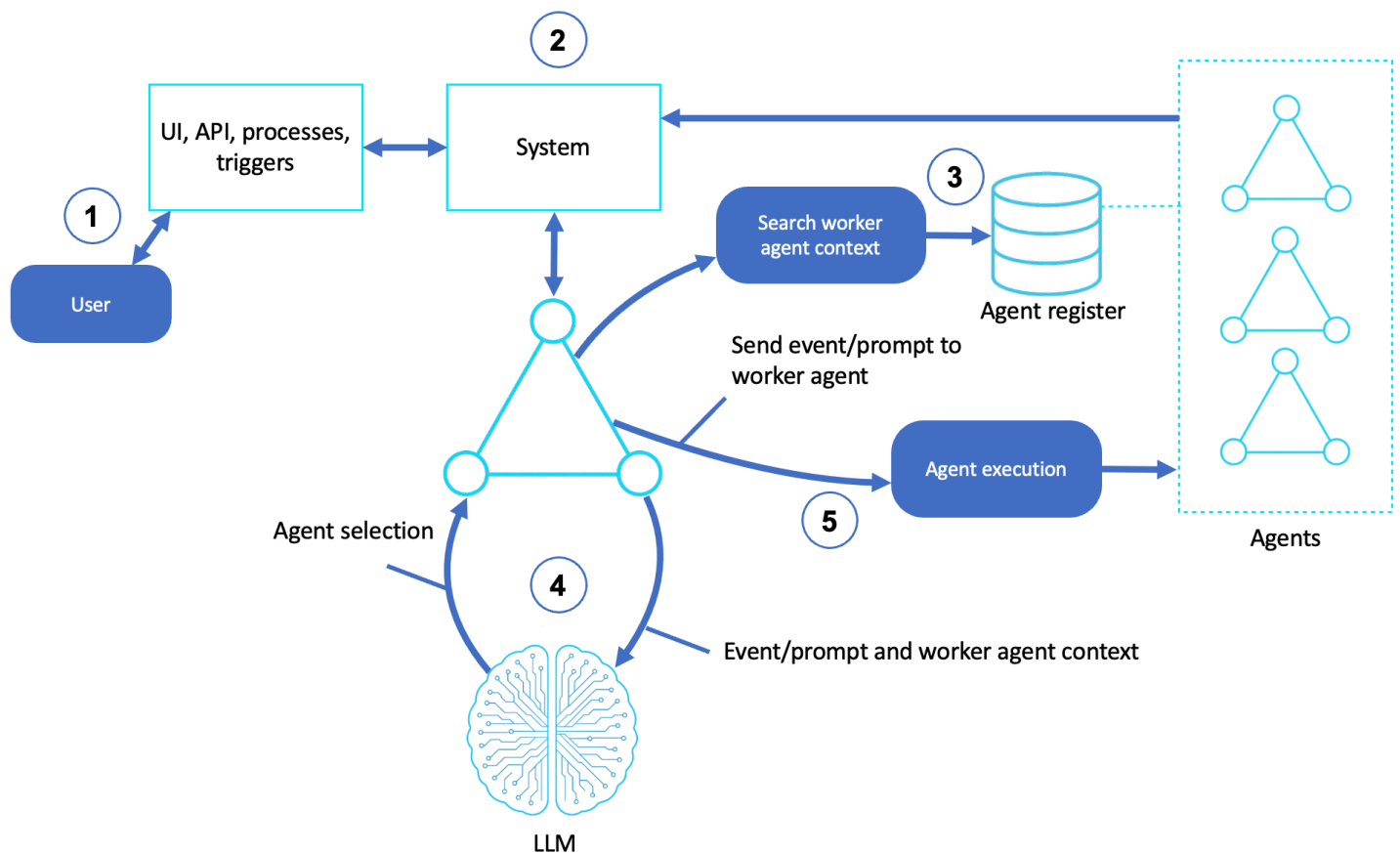
Agen orkestrasi alur kerja

Agen orkestrasi alur kerja mengelola dan mengoordinasikan tugas, proses, dan layanan multistep di seluruh sistem terdistribusi. Alih-alih bernalar dan bertindak secara terpisah, agen ini mendelegasikan pekerjaan ke subagen atau sistem lain, mempertahankan konteks eksekusi, dan beradaptasi berdasarkan hasil antara.

Agen ini adalah bagian mendasar dari arus otomatisasi. Mereka sangat berguna saat menangani tugas yang berjalan lama, komposisi multi-agen, dan integrasi lintas domain di mana berbagai agen dan alat harus dipanggil secara berurutan atau kondisional.

Arsitektur

Agen orkestrasi alur kerja ditunjukkan pada diagram berikut:



Deskripsi

1. Menerima masukan pengguna

- Pengguna (atau pemicu eksternal) memulai tugas melalui UI, API, atau peristiwa sistem.

2. Menangani peristiwa sistem

- Komponen sistem menerima permintaan dan memancarkan peristiwa atau perintah yang membutuhkan orkestrasi.

3. Mengambil konteks

- Agen alur kerja menanyakan basis pengetahuan dan pendaftar agen untuk menemukan agen pekerja yang tepat untuk tugas berdasarkan metadata, domain, dan tingkat keberhasilan sebelumnya.

4. Memilih agen LLM

- LLM membantu memilih agen atau rencana alur kerja terbaik dengan menganalisis deskripsi tugas dan opsi yang tersedia.
- Ini juga dapat merumuskan permintaan khusus tugas untuk dikirim ke agen yang dipilih.

5. Mendelegasikan dan mengeksekusi

- Agen pekerja yang dipilih menerima acara atau prompt dan mulai menjalankan perintah.
- Ini dapat melacak status eksekusi, mencoba lagi kegagalan, dan meneruskan hasil perantara ke agen berikutnya dalam urutan.

Kemampuan

- Komposisi agen (misalnya, supervisor, agen kolaborator, dan alat)
- Event-driven atau eksekusi terjadwal
- Memori dan melacak status dari waktu ke waktu
- Orkestrasi tugas hierarkis atau paralel (sinkron dibandingkan dengan alur kerja asinkron)
- Pemilihan dan rantai agen dinamis

Kasus penggunaan umum

- Otomatisasi multistep (misalnya, penyerapan dan pelaporan data)
- Perutean dan eskalasi layanan pelanggan (misalnya, agen-sebagai-koordinator)
- Agen AI berkoordinasi dengan manusia dan bot dalam lingkaran yang sama
- Mengotomatiskan proses perusahaan menggunakan logika LLM-powered
- Sistem hibrida menggabungkan agen AI dan alat orkestrasi tradisional

Panduan implementasi

Anda dapat membangun pola ini menggunakan alat-alat berikut dan Layanan AWS:

- Amazon Bedrock untuk penalaran dan pemilihan agen
- AWS Step Functions atau Amazon EventBridge untuk komposisi alur kerja
- AWS Lambda sebagai unit eksekusi atau pelari tugas
- Amazon DynamoDB, Amazon Simple Storage Service (Amazon S3) Simple Storage Service, atau Amazon RDS untuk melacak status dan hasil
- AWS AppFabric atau Amazon AppFlow untuk koordinasi lintas sistem
- (Opsional) Gunakan agen yang SageMaker dijalankan Amazon untuk meng-host agen pekerja khusus domain

Ringkasan

Agan alur kerja mengoordinasikan, beradaptasi, dan menyelaraskan tujuan di lingkungan multi-agen. Ini berarti bahwa agen AI dapat berkolaborasi, beradaptasi dengan kondisi runtime, dan memberikan hasil yang kompleks melalui alur kerja modular yang dapat dijelaskan.

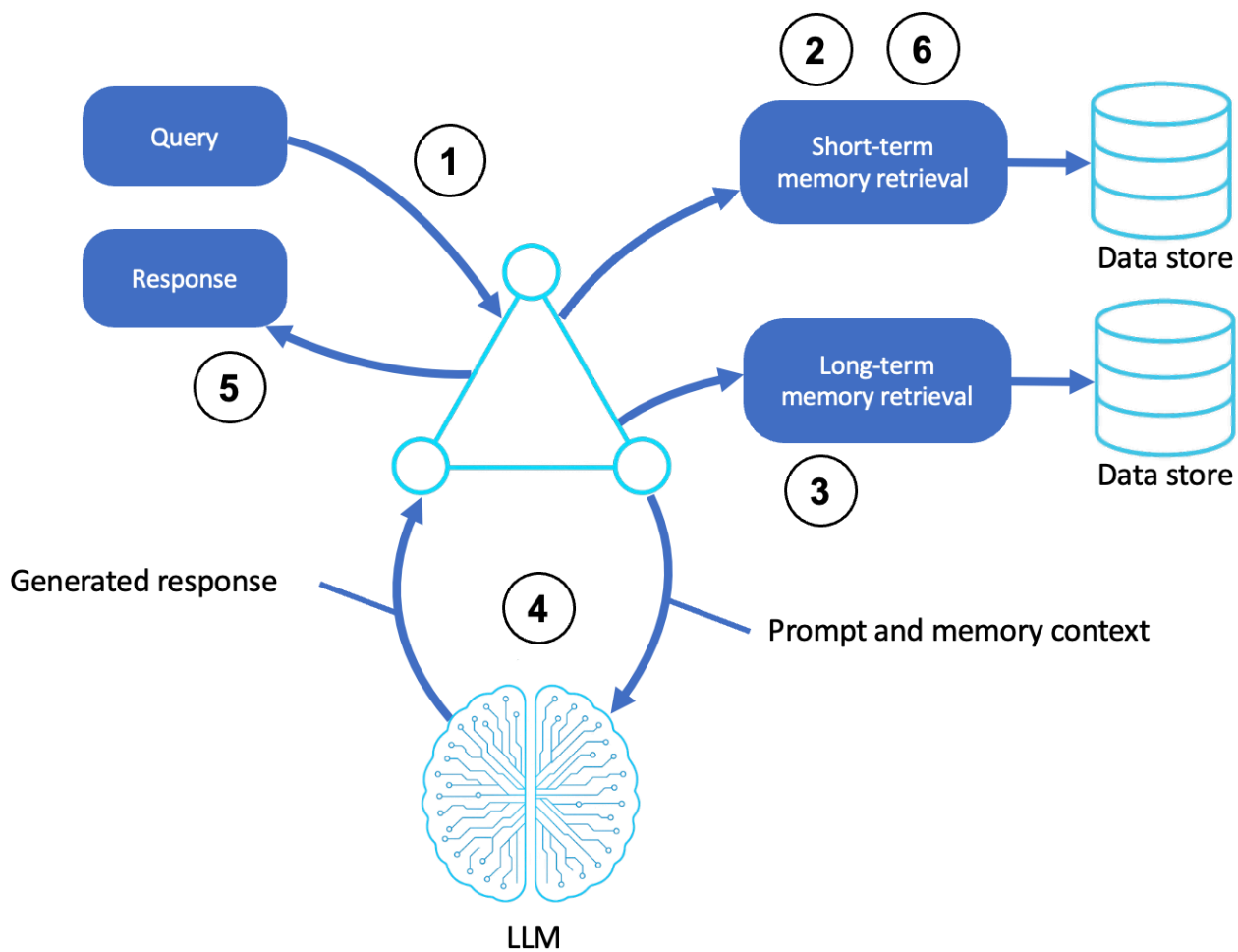
Memory-augmented agen

Memory-augmented agen ditingkatkan dengan kemampuan untuk menyimpan, mengambil, dan bernalar menggunakan memori jangka pendek dan jangka panjang. Hal ini memungkinkan mereka untuk mempertahankan konteks di berbagai tugas, sesi, dan interaksi, yang menghasilkan respons yang lebih koheren, personal, dan strategis.

Tidak seperti agen tanpa kewarganegaraan, agen tambahan memori beradaptasi dengan mereferensikan data historis, belajar dari hasil sebelumnya, dan membuat keputusan yang sesuai dengan tujuan, preferensi, dan lingkungan pengguna.

Arsitektur

Agan tambahan memori ditunjukkan pada diagram berikut:



Deskripsi

1. Menerima masukan atau acara

- Agen menerima permintaan pengguna atau peristiwa sistem. Ini mungkin teks, pemicu API, atau perubahan lingkungan.

2. Mengambil memori jangka pendek

- Agen mengambil riwayat percakapan terbaru, konteks tugas, atau status sistem yang relevan dengan sesi atau alur kerja.

3. Mengambil memori jangka panjang

- Agen menanyakan memori jangka panjang (misalnya, database vektor dan penyimpanan nilai kunci) untuk wawasan historis, seperti berikut ini:
 - Preferensi pengguna
 - Keputusan dan hasil masa lalu

- Konsep, ringkasan, atau pengalaman yang dipelajari

4. Alasan melalui LLM

- Konteks memori tertanam ke dalam prompt LLM, memungkinkan agen untuk bernalar berdasarkan input saat ini dan pengetahuan sebelumnya.

5. Menghasilkan output

- Agen menghasilkan respons, rencana, atau tindakan sadar kontekstual yang dipersonalisasi sesuai dengan riwayat tugas dan input pengguna.

6. Memperbarui memori

- Informasi baru, seperti sasaran yang diperbarui, sinyal keberhasilan dan kegagalan, dan respons terstruktur, disimpan untuk tugas-tugas masa depan.

Kemampuan

- Kontinuitas sesi di seluruh percakapan atau acara
- Kegigihan tujuan dari waktu ke waktu
- Kesadaran kontekstual berdasarkan keadaan yang berkembang
- Kemampuan beradaptasi diinformasikan oleh keberhasilan dan kegagalan sebelumnya
- Personalisasi selaras dengan preferensi dan riwayat pengguna

Kasus penggunaan umum

- Copilot percakapan yang mengingat preferensi pengguna
- Agen pengkodean yang melacak perubahan basis kode
- Agen alur kerja yang beradaptasi sesuai dengan riwayat tugas
- Kembar digital yang berkembang dari pengetahuan sistem
- Agen penelitian yang menghindari pengambilan berlebihan

Menerapkan agen yang diperbesar memori

Gunakan alat-alat berikut dan Layanan AWS untuk agen tambahan memori:

Lapisan memori

Layanan AWS

Tujuan

Short-term	Amazon DynamoDB, Redis, konteks Batuan Dasar Amazon	Pengambilan cepat status interaksi terbaru
Long-term (terstruktur)	Amazon Aurora, Amazon DynamoDB, Amazon Neptune	Fakta, hubungan, dan log
Long-term (semantik)	OpenSearch, PostgreSQL, Biji pinus	Embedding-based pengambilan (yaitu, RAG)
Penyimpanan	Amazon S3	Menyimpan transkrip, memori terstruktur, dan file
Orkestrasi	AWS Lambda atau AWS Step Functions	Mengelola injeksi memori dan memperbarui siklus hidup
Penalaran	Amazon Bedrock	Anthropic Claude atau Mistral dengan petunjuk memori

Menerapkan dorongan yang disuntikkan memori

Untuk mengintegrasikan memori ke dalam penalaran agen, gunakan kombinasi keadaan terstruktur dan injeksi konteks yang ditambah pengambilan:

- Sertakan status agen terbaru dan riwayat dialog terbaru sebagai input terstruktur saat membuat prompt untuk model bahasa, sehingga dapat beralasan dengan konteks penuh.
- Gunakan retrieval-augmented generation (RAG) untuk menarik dokumen atau fakta yang relevan dari memori jangka panjang.
- Meringkas rencana, konteks, dan interaksi sebelumnya untuk kompresi dan relevansi.
- Suntikkan modul memori eksternal, seperti penyimpanan vektor atau log terstruktur, selama inferensi untuk memandu pengambilan keputusan.

Ringkasan

Memory-augmented agen mempertahankan kontinuitas pikiran dengan belajar dari pengalaman dan mengingat konteks pengguna. Agen-agen ini melampaui kecerdasan reaktif dengan menggunakan

kolaborasi jangka panjang, personalisasi, dan penalaran strategis. Dalam hal AI agen, memori memungkinkan agen untuk berperilaku lebih seperti rekan digital adaptif dan kurang seperti alat stateless.

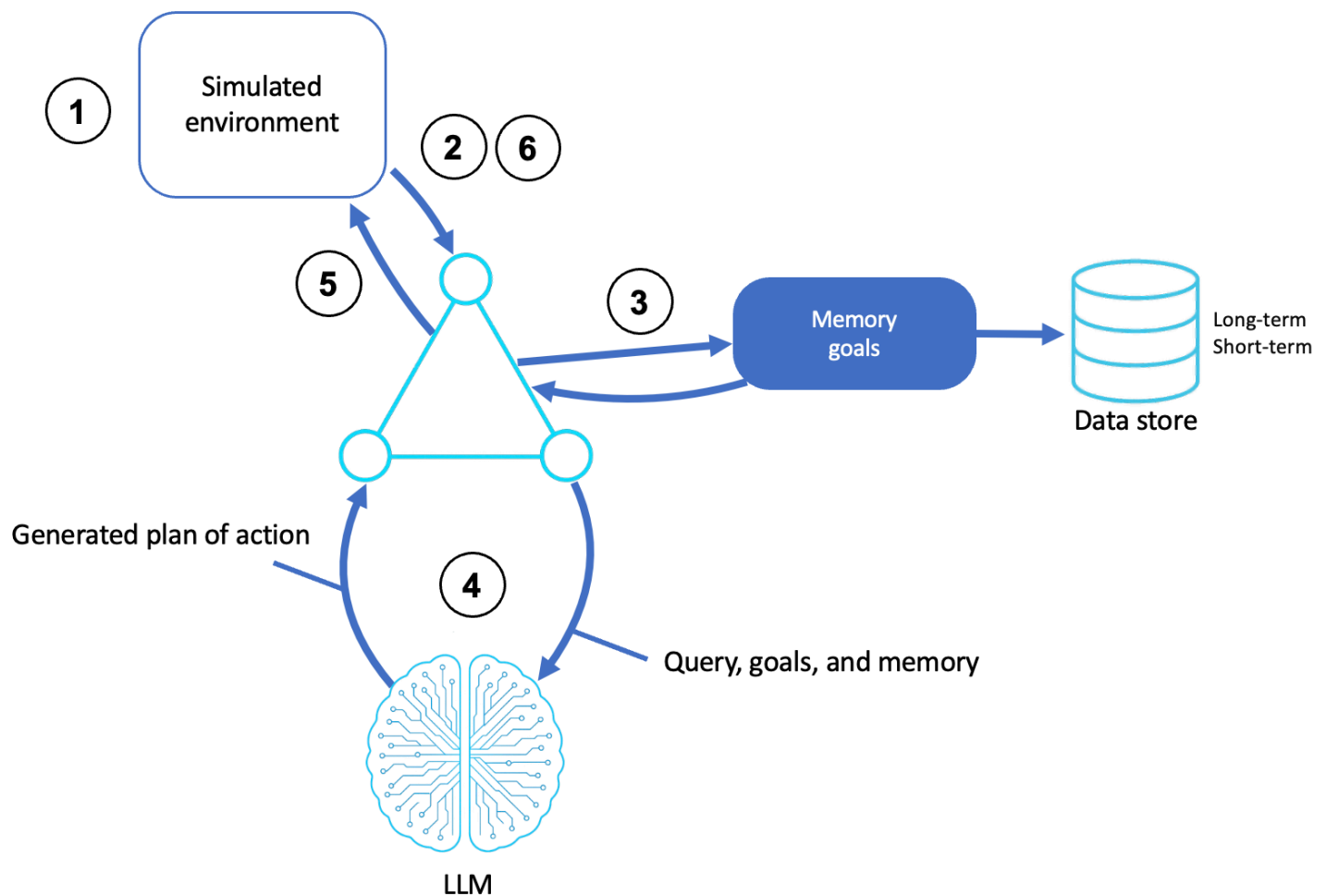
Agen simulasi dan test-bed

Agen simulasi dan test-bed beroperasi dalam lingkungan virtual atau terkontrol di mana mereka bernalar, bertindak, dan belajar. Agen ini mensimulasikan perilaku, memodelkan hasil, dan melatih strategi dalam pengaturan yang dapat diulang sebelum menerapkannya ke lingkungan dunia nyata.

Pola ini berguna untuk pengembangan berulang, pembelajaran penguatan (RL), evaluasi pengambilan keputusan otonom, dan pengujian perilaku yang muncul. Agen simulasi sering beroperasi dalam loop tertutup, menerima umpan balik dari lingkungan mereka dan menyesuaikan perilaku mereka sesuai, menjadikannya penting untuk tugas-tugas yang melibatkan penalaran spasial, kontrol waktu nyata, atau dinamika sistem yang kompleks.

Arsitektur

Diagram berikut menunjukkan simulasi atau agen test-bed:



Deskripsi

1. Memulai lingkungan

- Agen memulai lingkungan simulasi (misalnya, dunia 3D, mesin fisika, kotak pasir CLI, atau aliran data sintesis).
- Agen dimuat ke lingkungan dengan tugas awal, tujuan, atau kebijakan.

2. Merasakan agen

- Agen merasakan keadaan saat ini melalui telemetri simulasi (misalnya, emulasi sensor, kamera virtual, dan log terstruktur).

3. Mengambil tujuan dan memori

- Agen mengambil tujuan yang ditetapkan, instruksi skenario, atau tujuan kontekstual.
- Ini juga dapat mengambil memori sebelumnya, termasuk yang berikut:
 - Long-term strategi atau kebijakan

- Peta lingkungan atau kendala yang diketahui
- Keberhasilan atau kegagalan masa lalu dari simulasi serupa

4. Alasan dan rencana

- LLM menafsirkan keadaan simulasi, tujuan tugas, dan pengetahuan yang dipelajari.
- Ini menghasilkan rencana tindakan atau perintah kontrol.

5. Menjalankan tindakan simulasi

- Agen mengeksekusi rencana, memodifikasi status, menavigasi ruang, atau berinteraksi dengan entitas virtual.

6. Belajar

- Agen mengevaluasi hasil tindakan
- Tergantung pada konfigurasi agen, mungkin melakukan hal berikut:
 - Lakukan RL
 - Catat hasil untuk fine-tuning di masa mendatang
 - Sesuaikan strategi secara real time

Kemampuan

- Beroperasi dalam lingkungan sintetis atau virtual
- Mendukung pembelajaran coba-coba, penyempurnaan kebijakan, dan pemodelan sistem
- Low-risk pengujian untuk perilaku, penanganan kegagalan, dan kasus tepi
- Mengaktifkan analisis perilaku agen yang muncul dalam pengaturan multi-agen
- Mendukung kontrol loop tertutup dan eksplorasi manusia-in-the-loop

Kasus penggunaan umum

- Pembelajaran penguatan untuk robotika, drone, dan game
- Pelatihan kendaraan otonom di jalan virtual
- Simulasi UI atau CLI untuk DevOps skenario test-bed
- Eksperimen perilaku yang muncul dalam simulasi sosial
- Validasi keamanan logika keputusan sebelum produksi

Panduan implementasi

Anda dapat membuat simulasi dan agen test-bed menggunakan alat-alat berikut dan: Layanan AWS

Komponen	Layanan AWS	Tujuan
Lingkungan	Amazon ECS, Amazon EC2, atau simulator khusus di lab studio Amazon SageMaker	Jalankan dunia virtual (Gazebo, Unity, Unreal) atau CLI kotak pasir
Logika agen	Amazon Bedrock, Amazon SageMaker, atau AWS Lambda	LLM-based perencana atau agen RL
Lingkaran umpan balik	Pembelajaran SageMaker penguatan Amazon, Amazon CloudWatch, atau log khusus	Pelacakan hadiah, penilaian hasil, dan pencatatan perilaku
Memori dan replay	Amazon S3, Amazon DynamoDB, atau Amazon RDS	Status persisten, riwayat episode, atau data skenario
Visualisasi	CloudWatch Dasbor Amazon atau notebook Amazon SageMaker	Amati perubahan kebijakan, hasil, dan metrik pelatihan

Berikut ini adalah aplikasi tambahan:

- [AWS SimSpace Weaver](#) untuk simulasi spasial skala besar
- [AWS IoT Core](#) untuk menguji perangkat bayangan
- [SageMaker Eksperimen Amazon](#) untuk evaluasi dan perbandingan agen

Ringkasan

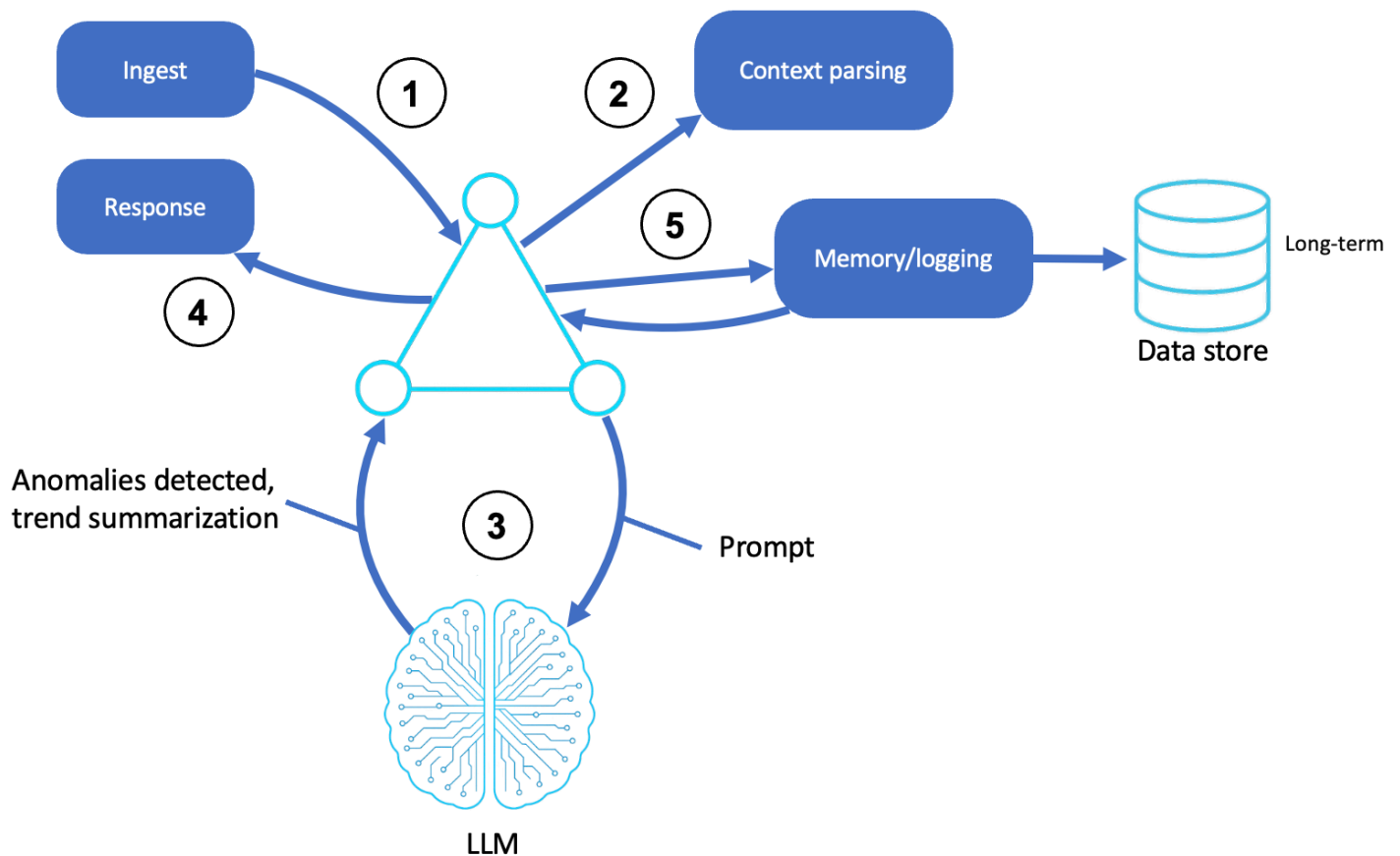
Agan simulasi dan test-bed adalah untuk eksplorasi terstruktur sebelum dikerahkan ke sistem produksi. Gunakan agen ini untuk melatih kebijakan navigasi otonom, menguji proses bisnis di lingkungan sintesis, dan mengevaluasi kawanan untuk pola koordinasi.

Pengamat dan agen pemantauan

Pengamat dan agen pemantauan secara pasif mengamati sistem, lingkungan, dan interaksi untuk mendeteksi pola, menghasilkan wawasan, dan memicu tindakan. Sebagai pengamat cerdas, mereka meningkatkan peringatan, diagnostik, dan audit tanpa langsung memulai perilaku.

Agen-agen ini unggul di mana pemantauan tradisional tidak memiliki kemampuan beradaptasi atau penalaran, terutama untuk AI-in-the-loop pemantauan, deteksi anomali, pengawasan kepatuhan, dan intelijen keamanan. Agen pengamat adalah pendengar acara yang terus memantau telemetry sistem dan interaksi pengguna. Agen tergantung pada persepsi, interpretasi, dan eskalasi atau pelaporan bersyarat.

Arsitektur



Deskripsi

1. Telemetri telemetri

- Agen menerima masukan dari satu atau lebih sumber sistem, seperti berikut ini:

- Log (aplikasi, infrastruktur, keamanan)
- Metrik (kinerja, latensi, penggunaan)
- Acara (panggilan API, tindakan pengguna, data sensor)

2. Mengurai konteks

- Input mentah diuraikan, terstruktur, dan diperkaya dengan metadata, seperti stempel waktu, identitas aktor, status sistem, dan ID jejak.

3. Alasan menggunakan LLM

- Agen menggunakan LLM atau modul logika untuk menafsirkan input yang diurai dengan mengidentifikasi anomali, meringkas tren, dan berkorelasi di seluruh jejak terdistribusi atau jendela waktu.

4. Klasifikasi atau waspada

- Agen menentukan apakah perilaku yang diamati menjamin hal-hal berikut:
 - Peringatan atau eskalasi
 - Laporan atau pembaruan dasbor
 - Pemicu respons (misalnya, remediasi otomatis dan penegakan kebijakan)

5. Memori log atau loop umpan balik

- Agen menyimpan acara dan keputusan untuk pembelajaran jangka panjang, audit, atau referensi future untuk agen lain.

Kemampuan

- Pasif dan noninvasif (agen tidak langsung bertindak)
- Sangat skalabel dan asinkron
- AI-driven korelasi di seluruh sinyal berisik atau terdistribusi
- Mendukung audit, kepatuhan, dan wawasan waktu nyata
- Dapat memberi makan agen hilir atau alur kerja manusia

Kasus penggunaan umum

- AI-augmented observabilitas untuk layanan mikro dan API
- Pemantauan penyimpangan model, pelanggaran kebijakan, atau perilaku out-of-band
- Analisis aktivitas pelanggan atau ringkasan interaksi

- Agen peninjau kode yang memantau komit atau penerapan
- Pemantauan log keamanan atau kepatuhan menggunakan penalaran LLM

Panduan implementasi

Anda dapat membangun pengamat dan agen pemantauan menggunakan alat-alat berikut dan Layanan AWS:

Komponen	Layanan AWS	Tujuan
Konsumsi acara	Amazon EventBridge, CloudWatch Log Amazon, Amazon Kinesis, Amazon S3	Menelan telemetry terstruktur dan tidak terstruktur
Pemrosesan awal	AWS Lambda, AWS Glue, AWS Step Functions	Ubah data mentah menjadi petunjuk terstruktur
Mesin penalaran	Batuan Dasar Amazon, Amazon SageMaker, AWS Lambda	Menganalisis peristiwa, mengklasifikasikan perilaku, menghasilkan wawasan
Penyimpanan dan memori	Amazon S3, Amazon DynamoDB, OpenSearch	Pengamatan, ringkasan, dan output yang terus-menerus
Peringatan dan eskalasi	Amazon SNS,, AWS AppFabric Amazon EventBridge	Memicu sistem atau agen hilir

Berikut ini adalah aplikasi tambahan:

- [AWS Security Hub CSPM](#) untuk pemantauan log keamanan
- [Amazon Quick](#) untuk memvisualisasikan output agen

Ringkasan

Pengamat dan agen pemantauan melacak sistem dan perilaku secara real time. Mereka mendeteksi anomali, mengaudit keamanan, dan mengumpulkan intelijen operasi dengan mengidentifikasi pola

yang mungkin diabaikan oleh manusia atau aturan. Kemampuan ini membantu menciptakan sistem yang dapat beradaptasi dengan perubahan kondisi dan membuat keputusan berdasarkan analisis data yang komprehensif.

Multi-agent kolaborasi

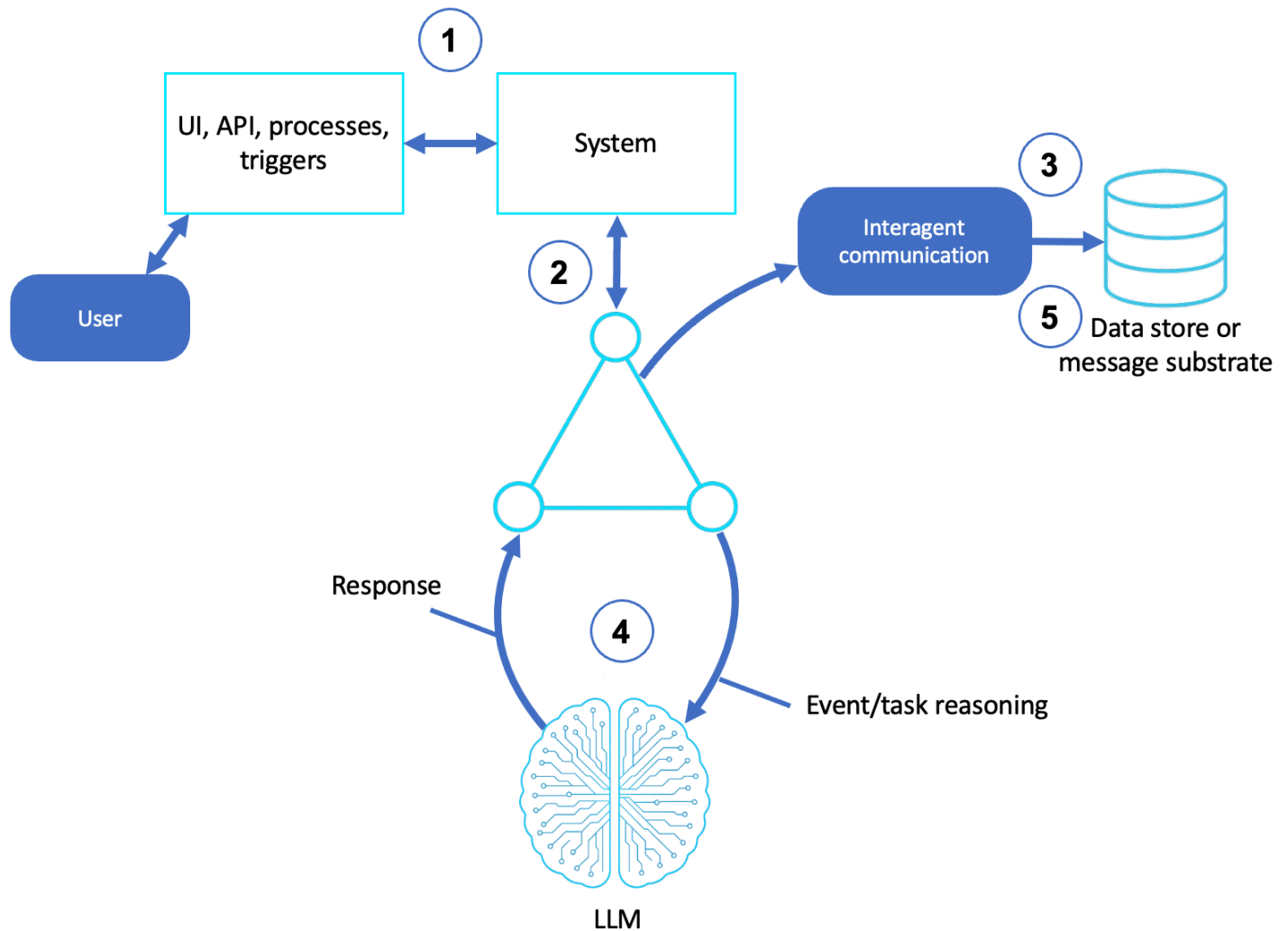
Multi-agent Kolaborasi mengacu pada pola di mana beberapa agen otonom, masing-masing dengan peran, spesialisasi, atau tujuan yang berbeda, bernegosiasi untuk menyelesaikan tugas-tugas yang kompleks. Agen ini dapat beroperasi secara independen atau dengan agen lain dengan berbagi informasi, membagi tanggung jawab, dan secara kolektif beralasan menuju suatu tujuan.

Pola ini berbeda dari agen alur kerja, yang secara terpusat mengoordinasikan dan mendelegasikan tugas ke agen bawahan dalam alur terstruktur. Sebaliknya, kolaborasi multi-agen menekankan koordinasi peer-to-peer atau emergent dengan memungkinkan adaptivitas, paralelisme, dan pembagian kognisi. Tabel berikut membandingkan kolaborasi multi-agen dengan agen alur kerja:

Fitur	Agen alur kerja	Tujuan
Pengendalian	Koordinator terpusat	Rekan-rekan yang terdesentralisasi, didistribusikan, atau berbasis peran
Interaksi	Satu agen mendelegasikan dan melacak eksekusi	Beberapa agen bernegosiasi, berbagi, dan beradaptasi
Desain	Urutan tugas yang telah ditentukan	Distribusi tugas yang muncul dan fleksibel
Koordinasi	Orkestrasi prosedural	Interaksi kooperatif atau kompetitif
Kasus penggunaan	Otomatisasi proses perusahaan	Penalaran kompleks, eksplorasi, dan strategi yang muncul

Arsitektur

Diagram berikut menunjukkan kolaborasi multi-agen:



Deskripsi

1. Memulai tugas

- Pengguna atau sistem memancarkan tujuan atau masalah tingkat tinggi.
- Agen “manajer” atau konteks awal mendefinisikan tujuan.

2. Menetapkan atau menemukan peran

- Agen menugaskan sendiri (logika simbolis atau penalaran) atau didelegasikan (broker acara) ke peran lain, seperti perencana, peneliti, pelaksana, kritikus, atau penjelasan.

3. Berkomunikasi dengan agen lain

- Agen berkomunikasi melalui memori bersama, antrian pesan, atau rantai cepat.
- Mereka mungkin berdebat, menanyakan, atau mengusulkan subtask satu sama lain.

4. Menggunakan penalaran khusus

- Setiap agen menggunakan model atau logika domainnya sendiri untuk memecahkan bagian masalahnya.
- Agen dapat menggunakan LLM dengan petunjuk dan memori khusus peran.

5. Mengkoordinasikan output atau tujuan

- Agen mensintesis kontribusi menjadi jawaban akhir, rencana, atau tindakan.
- (Opsional) Agen pengawas dapat memvalidasi atau meringkas output yang disintesis.

Kemampuan

- Peer-level agen dengan peran atau keterampilan khusus
- Perilaku yang muncul melalui komunikasi atau negosiasi
- Pemrosesan paralel masalah kompleks atau multifaset
- Mendukung musyawarah, koreksi diri, dan iterasi reflektif
- Model dinamika sosial, kolaborasi ilmiah, atau peran tim perusahaan

Kasus penggunaan umum

- Tim peneliti otonom (agen pencarian, peringkas, dan validator)
- Pengembangan perangkat lunak (perencana, pembuat kode, dan penguji)
- Pemodelan skenario bisnis (keuangan, kebijakan, dan kepatuhan)
- Negosiasi, penawaran, atau penalaran multipihak
- Tugas multimodal (gambar, teks, dan logika)

Panduan implementasi

Anda dapat membangun sistem multi-agen menggunakan alat-alat berikut dan Layanan AWS:

Komponen	Layanan AWS	Tujuan
Agan hosting	Batuan Dasar Amazon, Amazon SageMaker, AWS Lambda	Tuan rumah LLM-driven agen individu

Lapisan komunikasi	Amazon SQS, Amazon, EventBridge AWS AppFabric	Pesan dan koordinasi antar agen
Memori bersama	Amazon DynamoDB, Amazon S3, atau OpenSearch	Multi-agent memori atau sistem papan tulis
Lapisan orkestrasi	AWS Step Functions, AWS Lambda jaringan pipa	Logika kickoff, timeout, fallback, dan coba lagi
Identifikasi agen	Agen Amazon Bedrock (ditentukan peran), dan AWS AppConfig Amazon Bedrock converse API (agen di luar Amazon Bedrock)	Role-based alat atau pemanggilan agen dan penegakan batas
Interaksi yang muncul	EventBridge Saluran pipa Amazon atau pendaftar agen	Aktifkan perutean tugas dinamis atau eskalasi

Ringkasan

Multi-agent kolaborasi mendistribusikan tugas pemecahan masalah di seluruh agen modular yang digerakkan oleh peran. Tidak seperti orkestrasi alur kerja, pola kolaborasi menggunakan kecerdasan, ketahanan, dan skalabilitas yang muncul yang mencerminkan bagaimana manusia memecahkan masalah. Ini sangat berharga untuk domain terbuka, tugas kreatif, penalaran multimodal, dan lingkungan yang mendapat manfaat dari beragam perspektif.

Kesimpulan

Pola yang dibahas sebelumnya menggambarkan pendekatan dasar untuk implementasi AI agen di dunia nyata. Dari penalaran dasar hingga kecerdasan yang ditambah memori, setiap pola dikonfigurasi secara unik untuk persepsi, kognisi, dan tindakan yang didasarkan pada otonomi, asinkron, dan agensi.

Pola-pola ini berbagi kosakata dan cetak biru teknis untuk membangun sistem yang cerdas dan diarahkan pada tujuan. Apakah pola disematkan dalam antarmuka pengguna, diatur melalui layanan cloud, atau dikoordinasikan di seluruh tim agen, setiap pola dapat disesuaikan dan modular.

Takeaways

- Pola agen dapat dikomposisi — Sebagian besar agen dunia nyata memadukan dua atau lebih pola (misalnya, agen suara dengan penalaran dan memori berbasis alat).
- Desain agen bersifat kontekstual — Pilih pola berdasarkan permukaan interaksi, kompleksitas tugas, toleransi latensi, dan kendala khusus domain.
- AWSimplementasi asli dapat dicapai — Dengan Amazon Bedrock SageMaker, Amazon,, AWS Lambda, dan arsitektur berbasis peristiwa AWS Step Functions, setiap pola agen dapat dikirimkan dalam skala besar.

Alur kerja LLM

Dalam pola agen, kami mengeksplorasi pola agen AI yang umum, masing-masing dibangun di sekitar serangkaian kemampuan modular: persepsi, tindakan, pembelajaran, dan kognisi. Inti dari modul kognitif dalam banyak pola agen adalah model bahasa besar (LLM) yang mampu penalaran, perencanaan, dan pengambilan keputusan. Namun, menerapkan LLM saja tidak cukup untuk menghasilkan perilaku yang cerdas dan diarahkan pada tujuan.

Untuk melakukan tugas kompleks dengan andal, agen harus menanamkan LLM dalam alur kerja terstruktur, di mana kemampuan model ditambah dengan alat, memori, loop perencanaan, dan logika koordinasi. Alur kerja LLM ini memungkinkan agen untuk memecah tujuan, merutekan subtugas, memanggil layanan eksternal, merefleksikan hasil, dan berkoordinasi dengan agen lain.

Bab ini memperkenalkan pola desain inti untuk membangun modul LLM-driven kognitif yang kuat, dapat diperluas, dan cerdas, yang diatur di sekitar alur kerja yang dapat digunakan kembali.

Dalam bagian ini

- [Ikhtisar LLM-augmented kognisi](#)
- [Alur kerja untuk rantai cepat](#)
- [Alur kerja untuk perutean](#)
- [Alur kerja untuk paralelisasi](#)
- [Alur kerja untuk orkestrasi](#)
- [Alur kerja untuk evaluator dan loop refleksi-refine](#)
- [Kesimpulan](#)

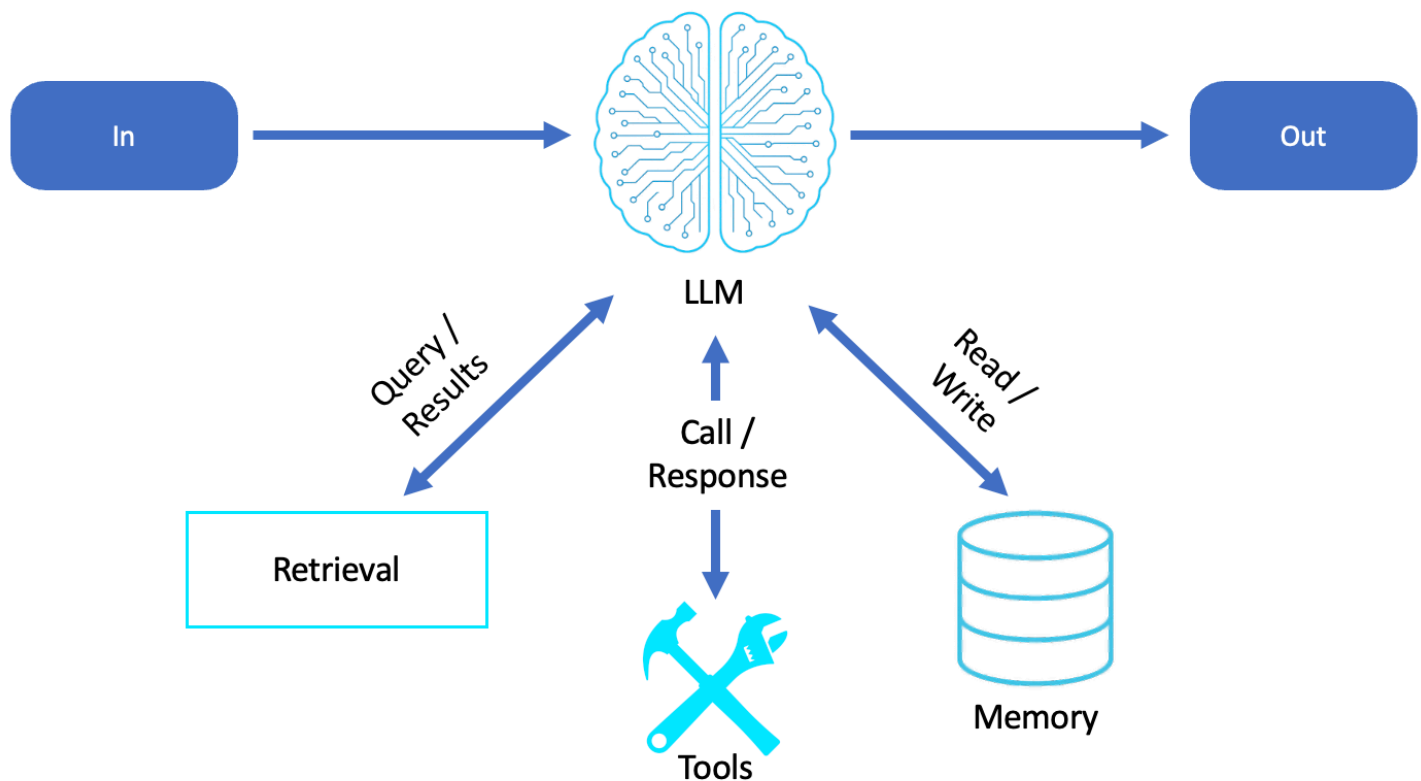
Ikhtisar LLM-augmented kognisi

Pada intinya, modul kognitif agen perangkat lunak dapat dilihat sebagai LLM yang dibungkus dengan augmentasi. Agen dapat menggunakan blok bangunan berikut untuk bernalar secara efektif dalam lingkungannya:

- **Meminta** — Membingkai input menggunakan konteks, instruksi, contoh, dan memori
- **Pengambilan** — Memberikan pengetahuan terkini atau spesifik domain ke prompt LLM melalui pencarian vektor atau memori semantik, misalnya, melalui retrieval-augmented generation (RAG)

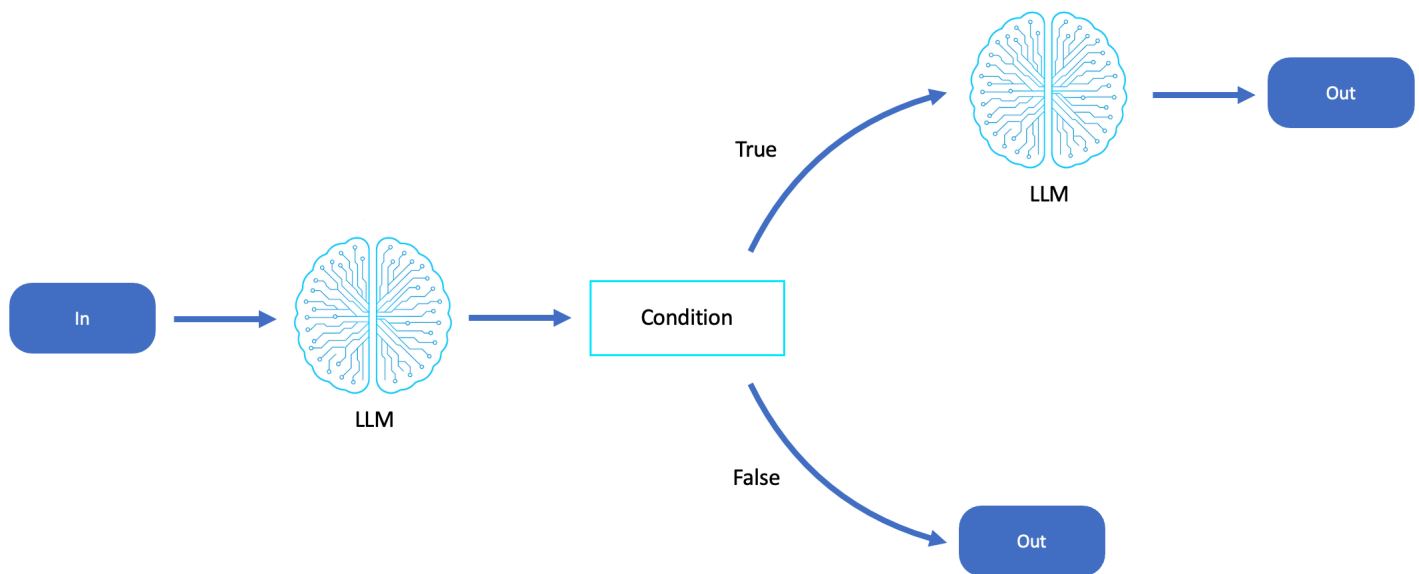
- Penggunaan alat - Mengaktifkan LLM untuk memanggil API atau memanggil fungsi untuk mengambil atau bertindak berdasarkan informasi
- Memori — Menggabungkan keadaan persisten atau berbasis sesi ke dalam loop penalaran, baik dengan menggunakan database terstruktur atau ringkasan kontekstual

Augmentasi ini terdiri dari alur kerja yang menentukan bagaimana LLM digunakan dari waktu ke waktu dan di seluruh tugas, mengubahnya dari mesin stateless menjadi agen penalaran dinamis.



Alur kerja untuk rantai cepat

Prompt chaining menguraikan tugas-tugas kompleks menjadi urutan langkah-langkah, di mana setiap langkah adalah pemanggilan LLM diskrit yang memproses atau membangun di atas output dari yang sebelumnya.



Alur kerja rantai prompt cocok untuk skenario di mana tugas dapat secara logis dibagi menjadi langkah-langkah penalaran berurutan, dan di mana output perantara menginformasikan tahap berikutnya. Ini unggul dalam alur kerja yang membutuhkan pemikiran terstruktur, transformasi progresif, atau analisis berlapis, seperti tinjauan dokumen, pembuatan kode, ekstraksi pengetahuan, dan penyempurnaan konten.

Deskripsi

- Kompleksitas tugas melebihi jendela konteks atau kedalaman penalaran dari satu panggilan LLM.
- Output dari satu langkah (misalnya, analisis, ringkasan, atau perencanaan) menjadi masukan untuk keputusan tindak lanjut atau fase generasi.
- Anda memerlukan transparansi dan kontrol di seluruh tahap penalaran (misalnya, hasil antara yang dapat diaudit).
- Anda ingin memasukkan validasi eksternal, pemfilteran, atau logika pengayaan di antara langkah-langkah.
- Ini ideal untuk agen yang beroperasi dalam loop penalaran gaya pipa, seperti agen penelitian, asisten editorial, sistem perencanaan, dan kopilot multistage.

Kemampuan

- Rantai linier atau bercabang dari panggilan LLM
- Hasil antara diteruskan sebagai input terstruktur atau disematkan ke dalam petunjuk tindak lanjut

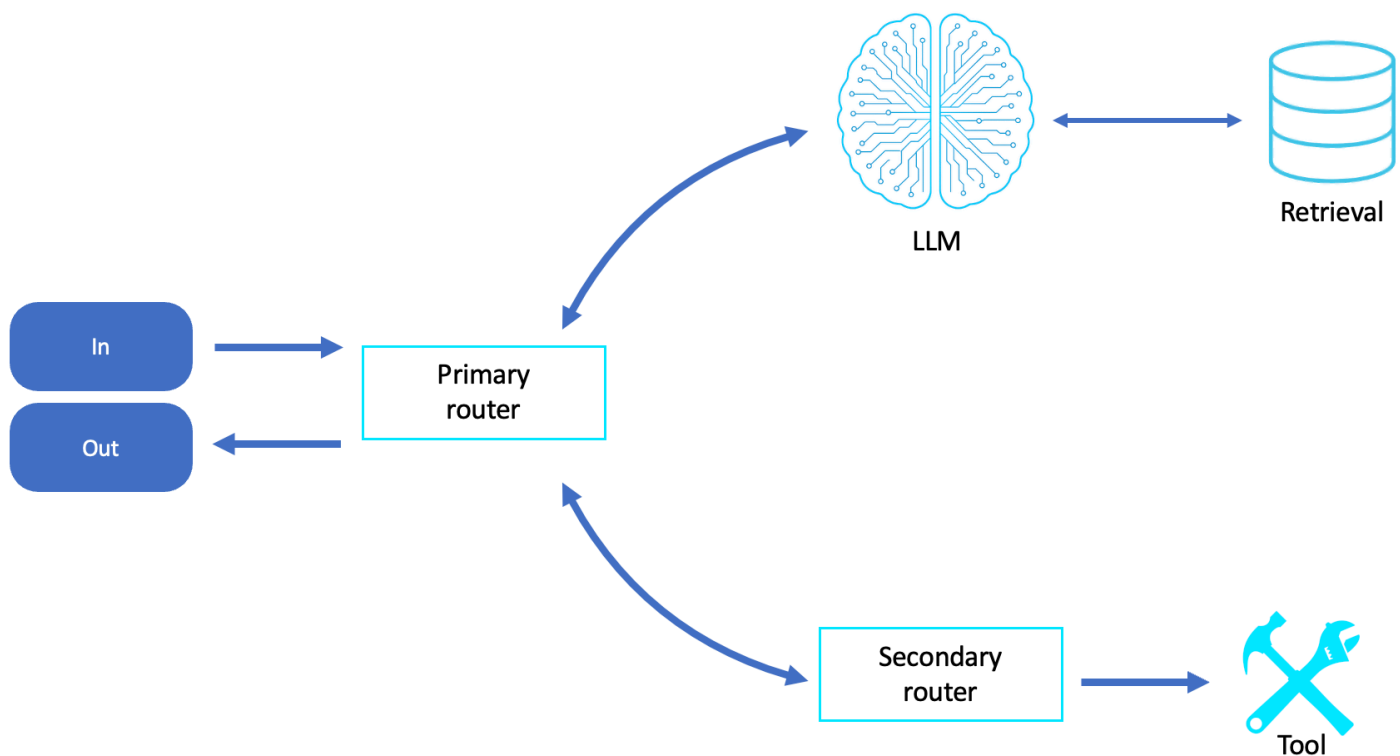
- Dapat diatur dengan,, atau pelari AWS Step Functions khusus AWS Lambda agen

Kasus penggunaan umum

- Tugas penalaran multistep (misalnya, “meringkas penulisan ulang kritik”)
- Asisten peneliti mensintesis keluaran berlapis (misalnya, “pertanyaan jawaban fakta ekstrak pencarian”)
- Pipa pembuatan kode (“hasilkan rencana tulis kode uji kode jelaskan output”)

Alur kerja untuk perutean

Dalam pola perutean, pengklasifikasi atau agen router menggunakan LLM untuk menafsirkan maksud atau kategori kueri, lalu merutekan input ke tugas atau agen hilir khusus.



Alur kerja perutean digunakan dalam skenario di mana agen harus mengklasifikasikan intent input, tipe tugas, atau domain dengan cepat, lalu mendelegasikan permintaan ke subagent, alat, atau alur kerja khusus. Ini sangat berguna dalam agen kapabilitas, seperti yang berfungsi sebagai asisten umum, pintu depan ke fungsi perusahaan, atau antarmuka AI yang menghadap pengguna yang menjangkau domain.

Routing sangat efektif ketika:

- Triaging permintaan di berbagai tugas (misalnya, pencarian, ringkasan, pemesanan, perhitungan).
- Input harus diproses sebelumnya atau dinormalisasi sebelum memasuki alur kerja yang lebih khusus.
- Jenis input yang berbeda (misalnya, gambar vs. teks, kueri terstruktur vs. tidak terstruktur) memerlukan penanganan khusus.
- Agen bertindak sebagai switchboard percakapan, mendelegasikan tugas ke agen khusus atau layanan mikro.
- Alur kerja ini umum terjadi pada kopilot khusus domain, bot dukungan pelanggan, router layanan perusahaan, dan agen multimodal, di mana pengiriman cerdas menentukan kualitas dan efisiensi perilaku agen.

Kemampuan

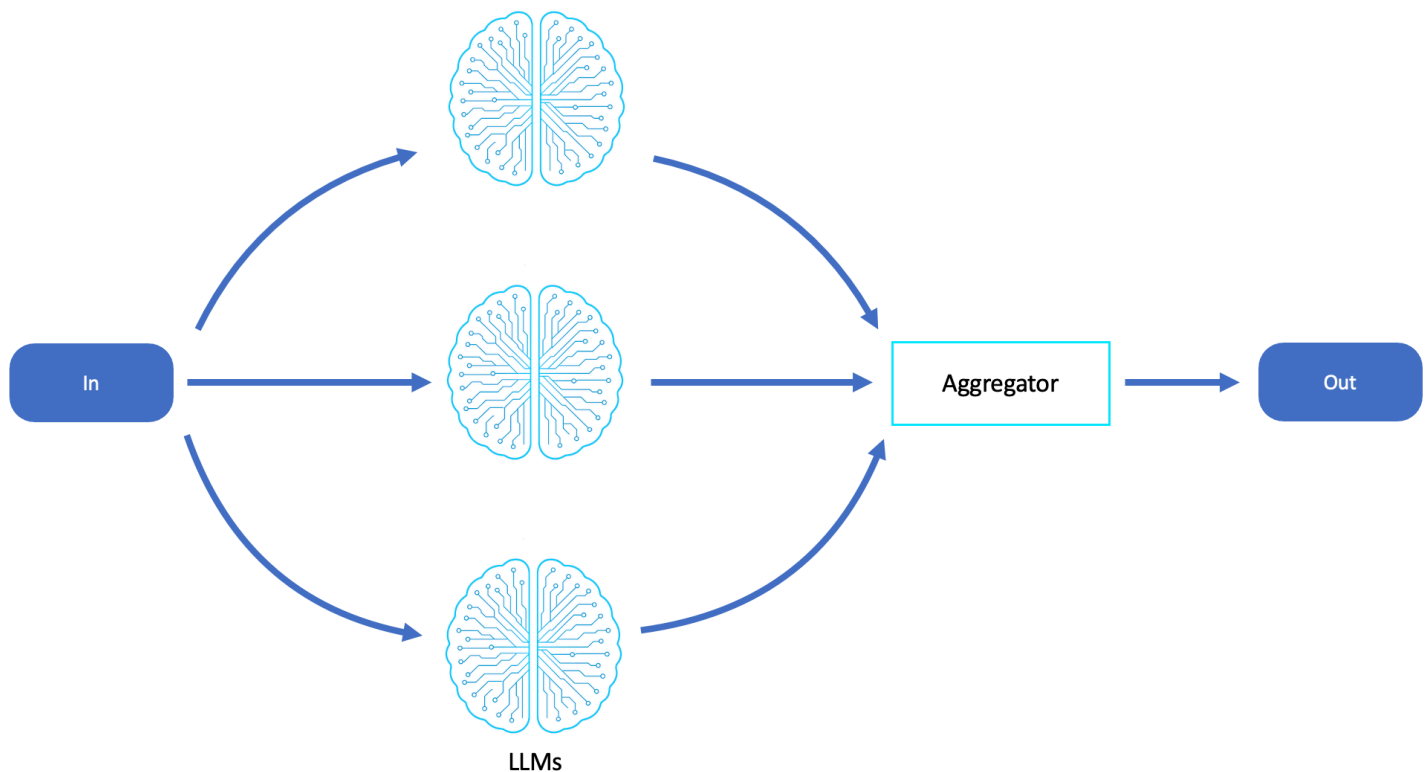
- LLM first-pass bertindak sebagai operator
- Rute dapat memanggil alur kerja yang berbeda atau bahkan pola agen lainnya
- Mendukung perluasan kemampuan modular

Kasus penggunaan umum

- Asisten multidomain (“apakah ini pertanyaan hukum, medis, atau keuangan?”)
- Pohon keputusan ditingkatkan dengan penalaran LLM
- Pemilihan alat dinamis (misalnya, pencarian vs. pembuatan kode)

Alur kerja untuk paralelisasi

Alur kerja ini melibatkan pemecahan tugas menjadi subtugas independen yang dapat ditangani secara bersamaan oleh beberapa panggilan atau agen LLM. Output kemudian digabungkan secara terprogram dan disintesis menjadi hasil.



Alur kerja Paralelisasi digunakan ketika tugas dapat dibagi menjadi subtask independen dan nonsekuensial yang dapat diproses secara bersamaan, secara signifikan meningkatkan efisiensi, throughput, dan skalabilitas. Ini sangat kuat dalam ruang masalah yang berat data, berorientasi batch, atau multiperspektif di mana agen harus menganalisis atau menghasilkan konten di beberapa input.

Paralelisasi sangat efektif ketika:

- Subtask tidak bergantung pada hasil antara satu sama lain, memungkinkan mereka berjalan secara paralel tanpa koordinasi.
- Tugas melibatkan pengulangan proses penalaran yang sama di banyak item (misalnya, meringkas beberapa dokumen atau mengevaluasi daftar opsi).
- Berbagai hipotesis atau perspektif dieksplorasi secara paralel untuk mempromosikan keragaman, kreativitas, atau ketahanan.
- Anda perlu mengurangi latensi untuk permintaan volume tinggi atau frekuensi tinggi melalui eksekusi LLM bersamaan.
- Alur kerja ini biasanya digunakan dalam agen pengolah dokumen, mesin survei atau perbandingan, ringkasan batch, brainstormer multi-agen, dan tugas klasifikasi atau pelabelan yang dapat diskalakan, terutama di mana penalaran paralel yang cepat merupakan keunggulan kinerja.

Kemampuan

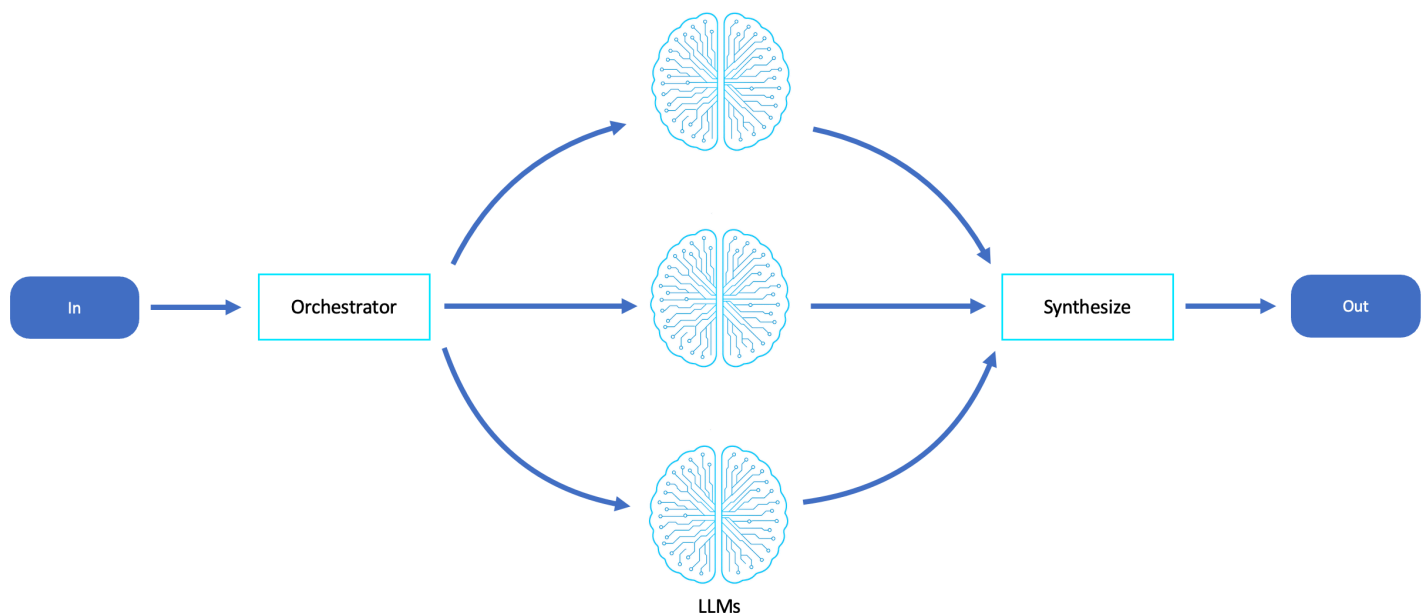
- Eksekusi paralel tugas LLM (dengan menggunakan AWS Lambda, AWS Fargate, atau status AWS Step Functions peta)
- Membutuhkan penyelarasan hasil, validasi, atau deduplikasi pada tahap sintesis
- Well-suited untuk loop agen stateless

Kasus penggunaan umum

- Menganalisis berbagai dokumen atau perspektif secara paralel
- Menghasilkan beragam konsep, ringkasan, atau rencana
- Mempercepat throughput di seluruh pekerjaan batch

Alur kerja untuk orkestrasi

Agen orkestrator pusat menggunakan LLM untuk merencanakan, menguraikan, dan mendelegasikan subtugas ke agen atau model pekerja khusus, masing-masing dengan peran atau keahlian domain tertentu. Ini mencerminkan struktur tim manusia dan mendukung perilaku yang muncul di berbagai agen.



Alur kerja orkestrasi sangat ideal untuk skenario yang kompleks, hierarkis, atau multidisiplin, yang membutuhkan dekomposisi terstruktur dan eksekusi khusus. Ini sangat cocok untuk tugas-tugas yang

membutuhkan pembagian kerja, di mana subkomponen tugas yang berbeda paling baik ditangani oleh agen dengan kemampuan, pengetahuan, atau perangkat yang berbeda.

Alur kerja ini sangat efektif ketika:

- Tugas dapat dibagi menjadi subtugas yang bervariasi dalam ruang lingkup, jenis, atau penalaran (misalnya, rencana, penelitian, implementasi, dan pengujian).
- LLM atau meta-agen harus mengoordinasikan agen lain, memantau kemajuan, dan mensintesis hasil.
- Anda ingin memodulasi tanggung jawab agen, memungkinkan skalabilitas, penggunaan kembali, dan penyetelan khusus.
- Sistem ini membutuhkan perilaku berbasis peran, meniru bagaimana tim manusia (misalnya, manajer proyek, pengembang, dan pengulas) beroperasi dalam kolaborasi.

Orkestrasi sangat ideal untuk agen perencanaan multiturn, kopilot pengembangan perangkat lunak, agen proses perusahaan, dan pelaksana proyek otonom. Ini sangat berguna ketika menerapkan sistem multi-agen yang memerlukan pemecahan tugas terpusat tetapi logika eksekusi terdistribusi, memungkinkan ekstensibilitas dan perilaku yang lebih dapat dijelaskan di seluruh lapisan agen.

Kemampuan

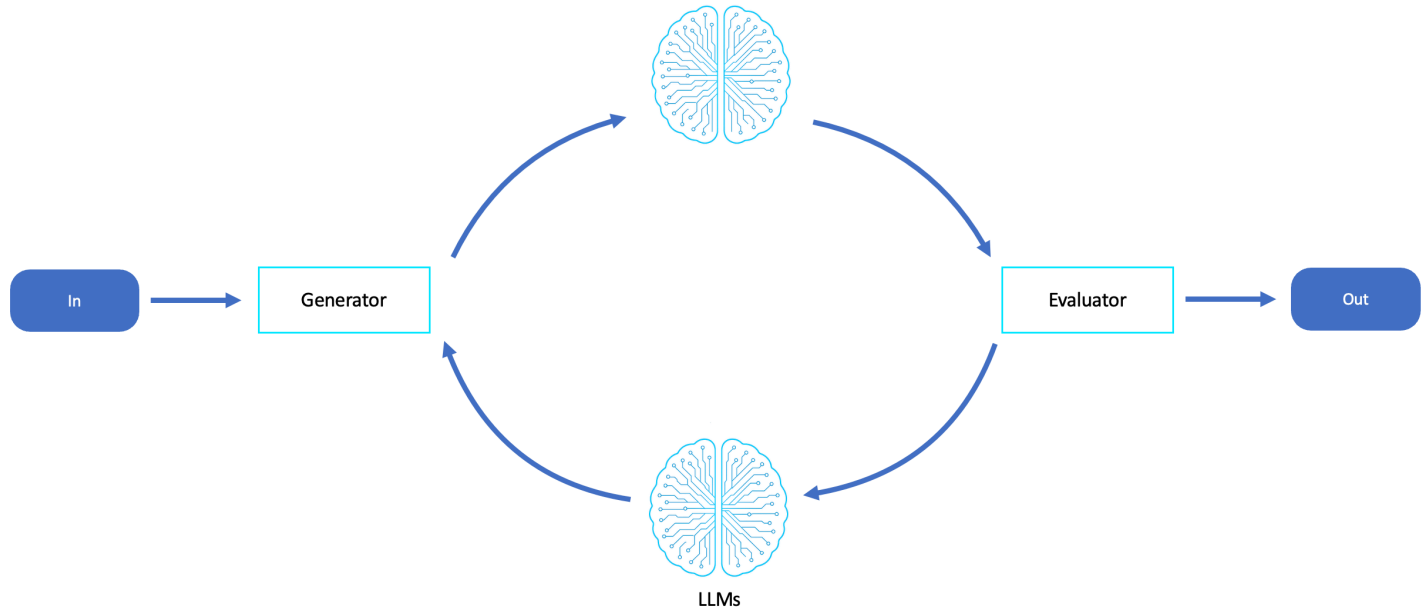
- Orkestrator melakukan meta-penalaran tujuan
- Agen pekerja dapat menyertakan akses alat, memori, atau permintaan khusus domain
- Bisa hierarkis (yaitu, delegasi tugas bertingkat)

Kasus penggunaan umum

- Manajer proyek, peneliti koordinasi, penulis, dan agen jaminan kualitas
- Coding copilot yang menggabungkan perencanaan, pelaksanaan, dan pengujian
- Agen yang mengawasi rantai alat atau pola akses API

Alur kerja untuk evaluator dan loop refleksi-refine

Alur kerja ini menyediakan loop umpan balik di mana satu LLM menghasilkan hasil, dan yang lain mengevaluasi atau mengkritik hasilnya. Ini mempromosikan refleksi diri, optimasi, dan peningkatan berulang.



Alur kerja evaluator sangat ideal untuk skenario di mana kualitas output, akurasi, dan penyelarasan penting dan di mana generasi single-pass tidak dapat diandalkan atau tidak memadai. Alur kerja ini unggul ketika agen harus mengkritik diri sendiri, mengulangi, dan menyempurnakan output mereka—baik untuk memenuhi standar kebenaran yang lebih tinggi atau untuk mengeksplorasi alternatif yang lebih baik berdasarkan umpan balik.

Alur kerja ini sangat efektif ketika:

- Output melibatkan metrik kualitas subjektif (misalnya, gaya, nada, dan keterbacaan) atau kriteria objektif (misalnya, kebenaran, keamanan, dan kinerja).
- Agen harus bernalar melalui trade-off, mengevaluasi kendala, atau mengoptimalkan menuju tujuan.
- Anda memerlukan redundansi bawaan dan jaminan kualitas, terutama dalam domain yang diatur, berhadapan dengan pelanggan, atau kreatif.
- Human-in-the-loop ulasan mahal atau tidak tersedia, dan validasi otonom diinginkan.

Alur kerja ini digunakan untuk pembuatan konten, sintesis dan peninjauan kode, penegakan kebijakan, pemeriksaan penyelarasan, penyetelan instruksi, dan pasca pemrosesan RAG. Ini juga

berguna untuk agen peningkatan diri, di mana umpan balik berkelanjutan membantu membentuk respons yang lebih baik dari waktu ke waktu untuk membangun loop keputusan otonom yang dapat dipercaya.

Kasus penggunaan umum

- Red-team agen dibandingkan dengan agen tim biru
- Agen yang menghasilkan, mengevaluasi, dan merevisi kode atau rencana
- Jaminan kualitas, deteksi halusinasi, dan penegakan gaya

Kemampuan

- Mendukung generasi dan evaluasi terpisah menggunakan model yang berbeda (misalnya, Claude untuk generasi dan Mistral untuk evaluasi)
- Umpan balik terstruktur dan digunakan untuk meminta output yang direvisi
- Mendukung beberapa iterasi atau ambang konvergensi

Kesimpulan

LLM menyediakan inti kognitif agen perangkat lunak modern, tetapi pemanggilan model mentah tidak cukup untuk mencapai kecerdasan yang terarah, kuat, dan dapat dikendalikan. Untuk beralih dari generasi keluaran ke penalaran terstruktur dan perilaku yang selaras dengan tujuan, LLM harus disematkan dalam pola alur kerja yang disengaja yang menentukan bagaimana model memproses input, mengelola konteks, dan mengoordinasikan tindakan.

Alur kerja LLM memperkenalkan fondasi untuk membangun modul kognitif agen:

- Rantai cepat memecah penalaran kompleks menjadi langkah-langkah modular yang dapat diaudit.
- Routing memungkinkan klasifikasi tugas cerdas dan delegasi yang ditargetkan.
- Paralelisasi mempercepat throughput dan mempromosikan beragam penalaran.
- Orkestrasi agen menyusun kolaborasi multi-agen melalui dekomposisi tugas dan eksekusi berbasis peran.
- Evaluator (reflect-refine loop) memungkinkan perbaikan diri, kontrol kualitas, dan pemeriksaan keselarasan.

Setiap alur kerja mewakili pola composable yang dapat disesuaikan dengan kebutuhan agen, kompleksitas tugas, dan harapan pengguna. Alur kerja ini tidak saling eksklusif. Mereka adalah blok bangunan yang sering digabungkan menjadi arsitektur hibrida yang mendukung penalaran dinamis, koordinasi multi-agen, dan keandalan tingkat perusahaan.

Saat Anda beralih ke babak berikutnya tentang pola alur kerja agen, alur kerja LLM ini akan muncul kembali sebagai struktur tertanam dalam sistem yang lebih besar, mendukung delegasi tujuan, orkestrasi alat, loop keputusan, dan otonomi siklus hidup. Menguasai alur kerja LLM ini sangat penting untuk merancang agen perangkat lunak yang tidak hanya memprediksi teks tetapi alasan, beradaptasi, dan bertindak dengan sengaja.

Pola alur kerja agen

Pola alur kerja agen mengintegrasikan agen perangkat lunak modular dengan alur kerja model bahasa besar (LLM) terstruktur, memungkinkan penalaran dan tindakan otonom. Meskipun terinspirasi oleh arsitektur tanpa server dan berbasis peristiwa tradisional, pola-pola ini mengubah logika inti dari kode statis ke LLM-augmented agen, memberikan peningkatan kemampuan beradaptasi dan pengambilan keputusan kontekstual. Evolusi ini mengubah arsitektur cloud konvensional dari sistem deterministik menjadi yang mampu interpretasi dinamis dan augmentasi cerdas, sambil mempertahankan prinsip-prinsip dasar skalabilitas dan responsif.

Dalam bagian ini

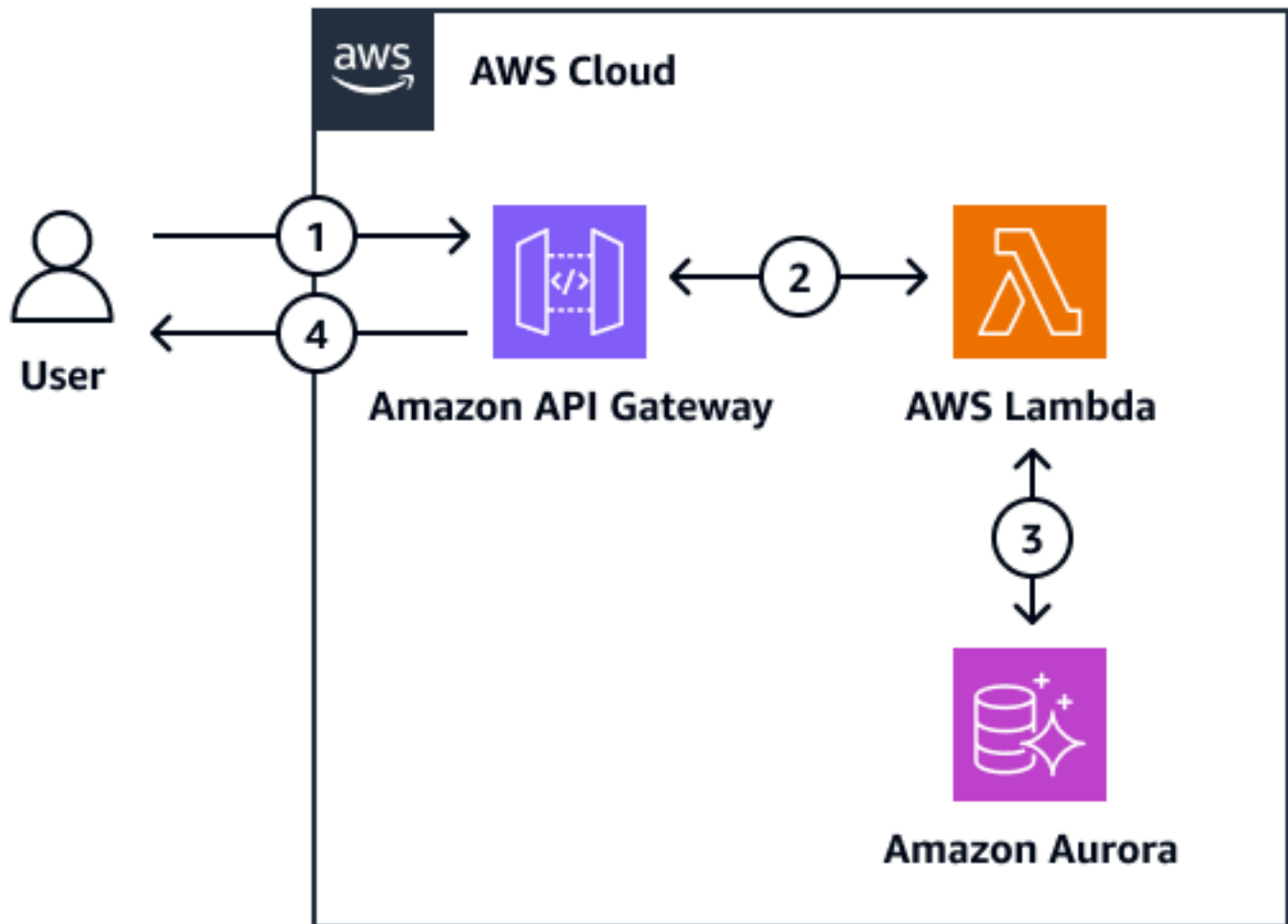
- [Dari sistem yang didorong oleh peristiwa hingga peningkatan kognisi](#)
- [Pola saga rantai yang cepat](#)
- [Merutekan pola pengiriman dinamis](#)
- [Paralelisasi dan pola scatter-gathering](#)
- [Pola orkestrasi Saga](#)
- [Evaluator refleksi-memperbaiki pola loop](#)
- [Merancang alur kerja agen pada AWS](#)
- [Kesimpulan](#)

Dari sistem yang didorong oleh peristiwa hingga peningkatan kognisi

Arsitektur cloud modern, terutama yang dibangun di atas prinsip tanpa server dan berbasis peristiwa, secara tradisional mengandalkan pola seperti perutean, fan-out, dan pengayaan untuk menciptakan sistem yang responsif dan dapat diskalakan. Sistem AI agen dibangun di atas fondasi ini sambil membingkainya LLM-augmented kembali di sekitar penalaran dan fleksibilitas kognitif. Pendekatan ini memungkinkan kemampuan pemecahan masalah dan otomatisasi yang lebih canggih, berpotensi merevolusi bagaimana tugas kompleks ditangani di lingkungan cloud.

Event-driven arsitektur

Diagram berikut menunjukkan sistem terdistribusi yang khas:

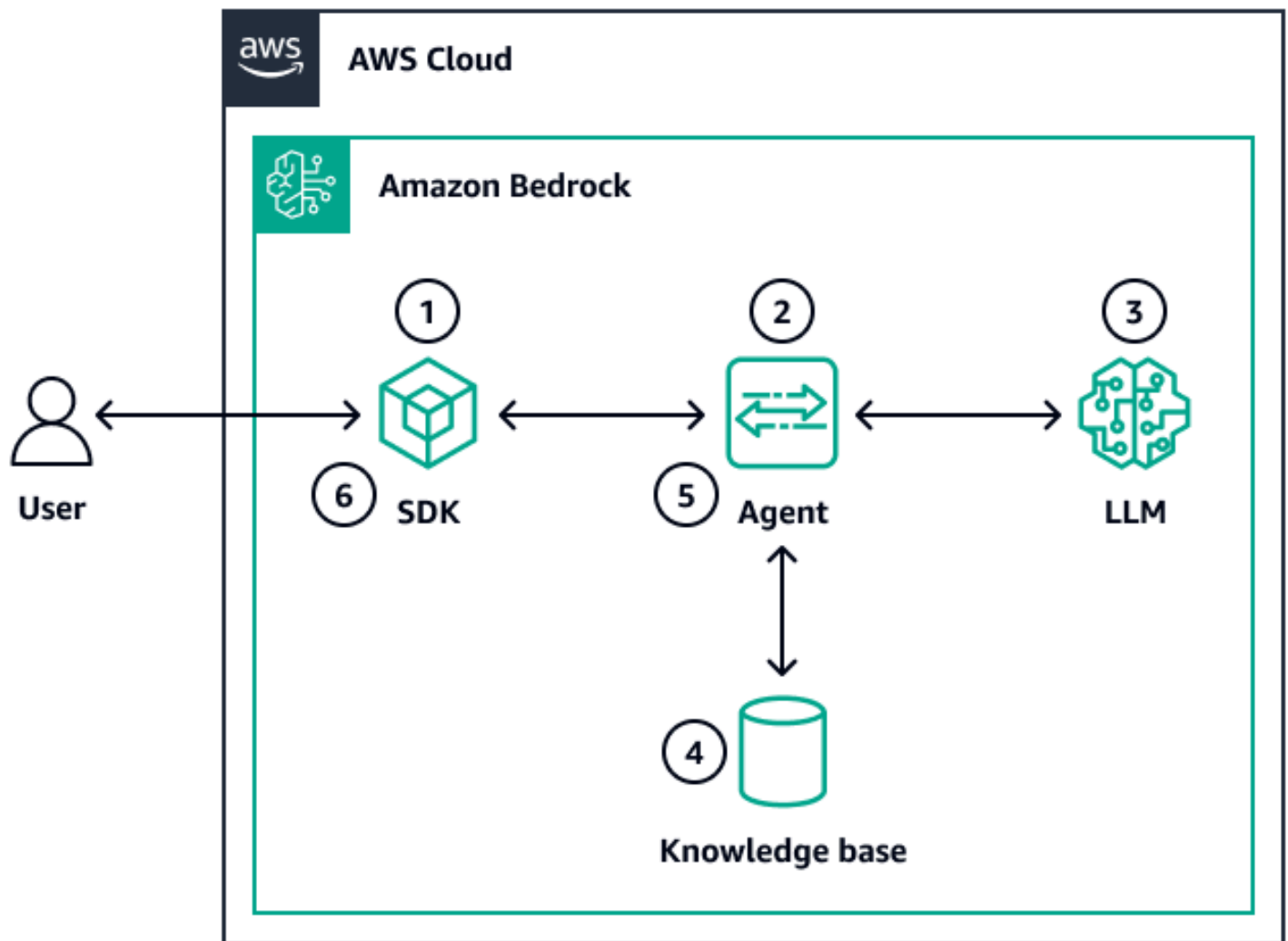


1. Pengguna mengirimkan permintaan ke Amazon API Gateway.
2. Amazon API Gateway merutekan permintaan ke suatu AWS Lambda fungsi.
3. AWS Lambda melakukan pengayaan data dengan menanyakan database Amazon Aurora
4. Amazon API Gateway mengembalikan payload yang diperkaya ke pemanggil.

Struktur ini dapat diandalkan dan dapat diskalakan, tetapi pada dasarnya statis. Aturan bisnis dan jalur logika harus dikodekan secara eksplisit, dan beradaptasi dengan perubahan konteks atau informasi yang tidak lengkap terbatas.

Cognition-augmented alur kerja

Arsitektur agen menambahkan augmentasi kognitif ke sistem yang digerakkan oleh peristiwa. Diagram berikut menunjukkan ekuivalen agen:



1. Pengguna mengirimkan kueri melalui panggilan SDK atau API.
2. Agen Amazon Bedrock menerima kueri.
3. Agen menafsirkan kueri dengan memanggil LLM
4. Agen melakukan pengayaan semantik dengan mencari basis pengetahuan Amazon Bedrock atau sumber data eksternal lainnya.
5. LLM mensintesis respons yang kaya konteks dan selaras dengan tujuan.
6. Sistem mengembalikan respons yang disintesis kepada pengguna.

Dalam alur ini, LLM menggunakan logika, memahami maksud, mengambil dan menggabungkan konteks yang relevan, dan kemudian memutuskan cara terbaik untuk merespons. Pola ini mencerminkan pola pengayaan tradisional, di mana pesan ditambah dengan data eksternal sebelum

diarahkan lebih lanjut. Namun, dalam sistem agen, pengayaan ini bukan pencarian statis. Sebaliknya, pengayaan bersifat dinamis, dibimbing secara semantik, dan didorong oleh tujuan.

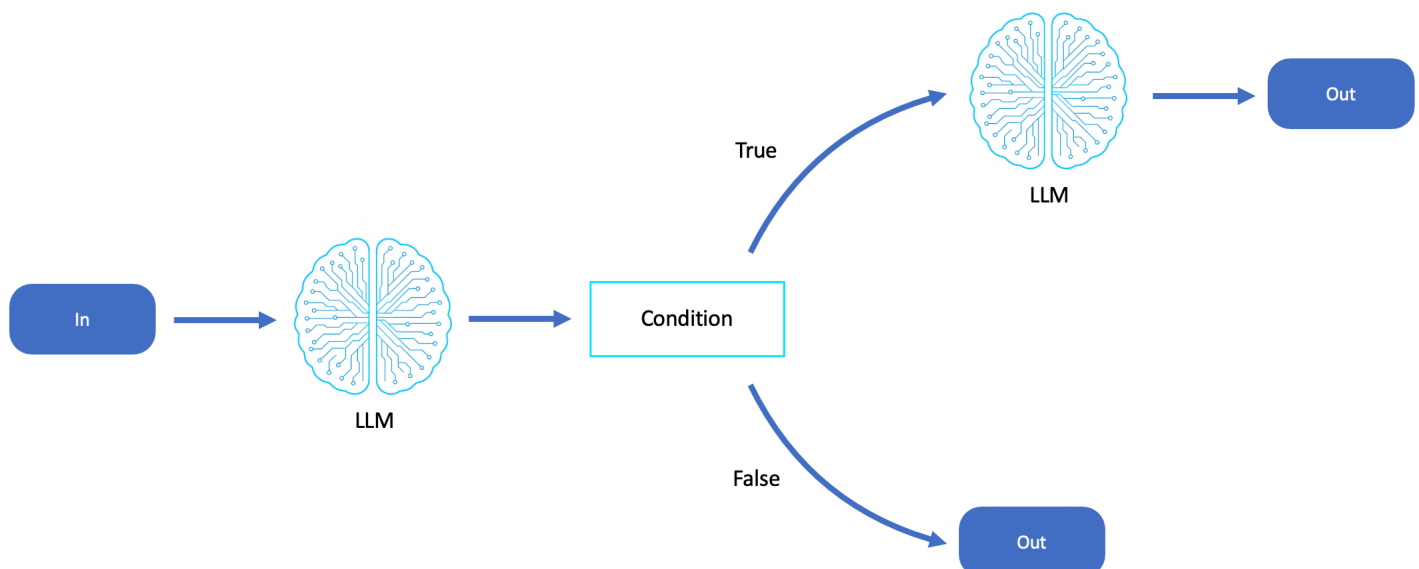
Wawasan inti

Setiap alur kerja LLM dapat dipetakan ke pola alur kerja agen, yang mencerminkan dan mengembangkan gaya arsitektur berbasis peristiwa tradisional. Blok bangunan dasar alur kerja agen adalah kemampuan untuk menambah konteks LLM dengan data, alat, dan memori. Ini menciptakan loop penalaran yang diinformasikan, adaptif, dan selaras dengan maksud pengguna. Di mana sistem tradisional memperkaya pesan dengan data pencarian, sistem agen memungkinkan perangkat lunak untuk bertindak kurang seperti skrip dan lebih seperti kolaborator cerdas.

Pola saga rantai yang cepat

Dengan menata ulang rantai prompt LLM sebagai kisah yang digerakkan oleh peristiwa, kami membuka model operasional baru: alur kerja menjadi terdistribusi, dapat dipulihkan, dan dikoordinasikan secara semantik di seluruh agen otonom. Setiap langkah prompt respons dibingkai ulang sebagai tugas atom, dipancarkan sebagai peristiwa, dikonsumsi oleh agen khusus, dan diperkaya dengan metadata kontekstual.

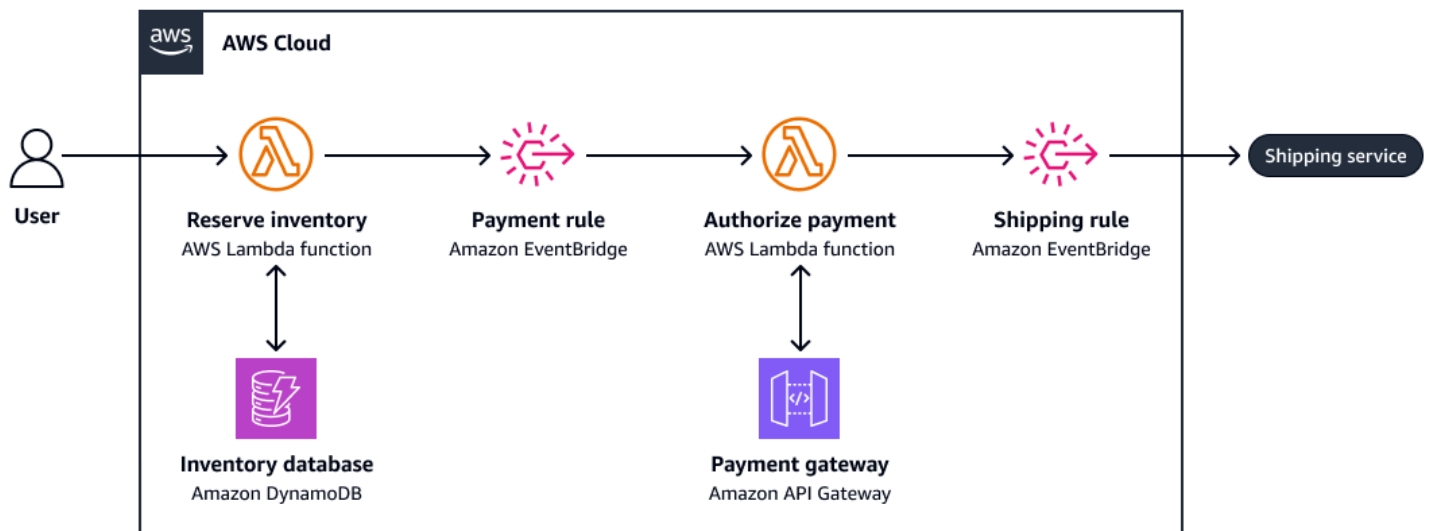
Diagram berikut adalah contoh rantai prompt LLM:



Koreografi Saga

Pola koreografi saga adalah pendekatan implementasi dalam sistem terdistribusi yang tidak memiliki koordinator pusat. Sebagai gantinya, setiap layanan atau komponen menerbitkan peristiwa yang memicu tindakan alur kerja berikutnya. Pola ini banyak digunakan dalam sistem terdistribusi untuk mengelola transaksi di berbagai layanan. Dalam sebuah kisah, sistem menjalankan serangkaian transaksi lokal yang terkoordinasi. Jika salah satu gagal, sistem memicu tindakan kompensasi untuk menjaga konsistensi.

Diagram berikut adalah contoh koreografi saga:



1. Persediaan cadangan
2. Otorisasi pembayaran
3. Buat pesanan pengiriman

Jika langkah 3 gagal, sistem akan meminta tindakan kompensasi (misalnya, membatalkan pembayaran atau merilis inventaris).

Pola ini sangat berharga dalam arsitektur yang digerakkan oleh peristiwa di mana layanan digabungkan secara longgar dan status harus diselesaikan secara konsisten dari waktu ke waktu, bahkan di hadapan kegagalan sebagian.

Pola rantai yang cepat

Prompt chaining menyerupai pola saga baik dalam struktur maupun tujuan. Ini mengeksekusi serangkaian langkah penalaran yang membangun secara berurutan sambil mempertahankan konteks dan memungkinkan untuk rollback dan revisi.

Koreografi agen

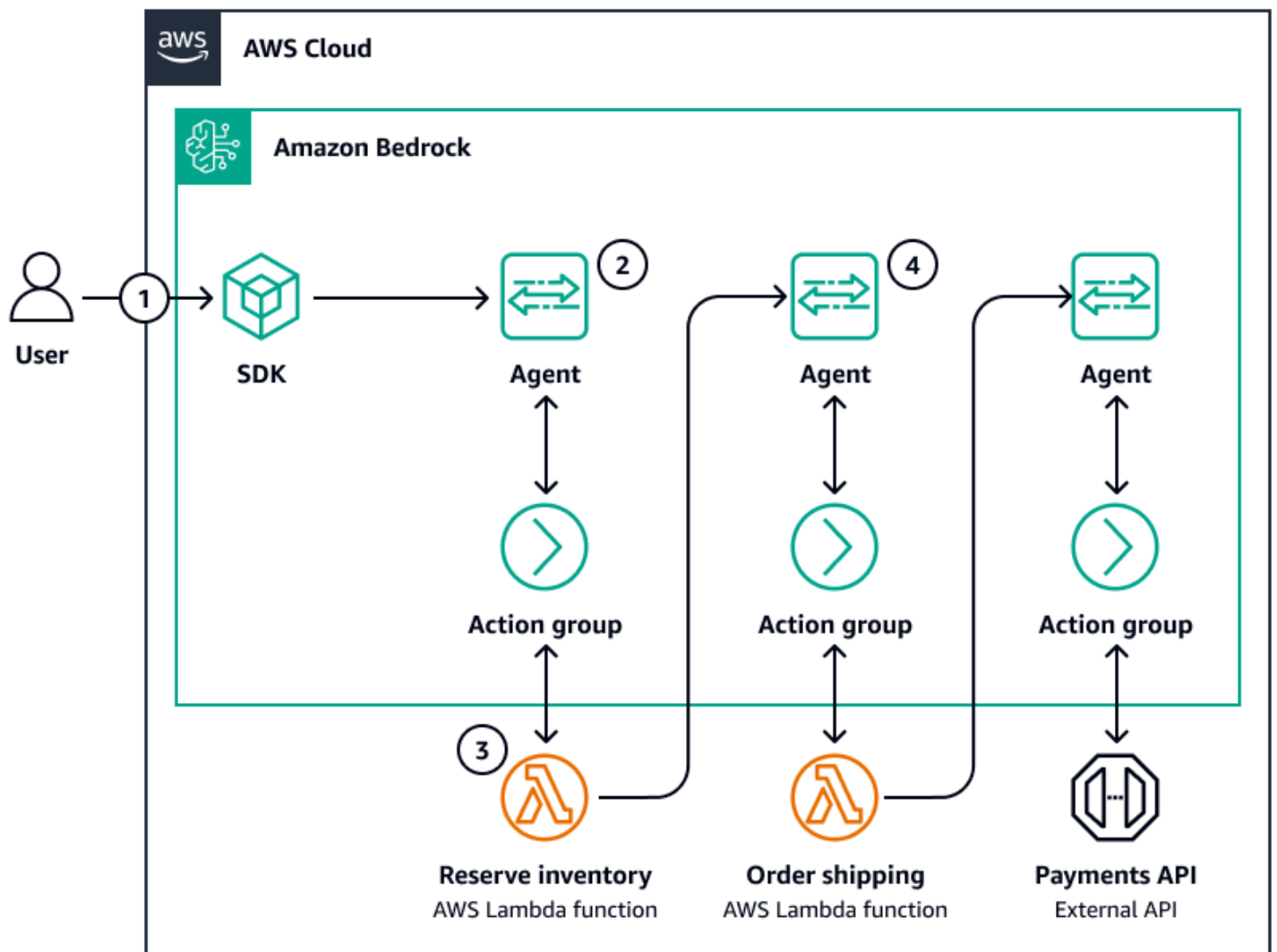
1. LLM menafsirkan kueri pengguna yang kompleks dan menghasilkan hipotesis
2. LLM menguraikan rencana untuk menyelesaikan tugas
3. LLM mengeksekusi subtugas (misalnya, dengan menggunakan panggilan alat atau mengambil pengetahuan)
4. LLM menyempurnakan output atau meninjau kembali langkah sebelumnya jika dianggap hasilnya tidak memuaskan

Jika hasil antara cacat, sistem dapat melakukan salah satu hal berikut:

- Coba lagi langkah-langkahnya menggunakan pendekatan yang berbeda
- Kembalikan ke prompt sebelumnya dan rencanakan ulang
- Gunakan loop evaluator (misalnya, dari pola evaluator-optimizer) untuk mendeteksi dan memperbaiki kegagalan

Seperti pola saga, rantai cepat memungkinkan untuk kemajuan sebagian dan mekanisme rollback. Hal ini terjadi melalui perbaikan berulang dan LLM-directed koreksi daripada melalui kompensasi transaksi database.

Diagram berikut adalah contoh koreografi agen:



1. Pengguna mengirimkan kueri melalui SDK.
2. Agen Amazon Bedrock mengatur penalaran melalui hal-hal berikut:
 - Interpretasi (LLM)
 - Perencanaan (LLM)
 - Eksekusi melalui alat atau basis pengetahuan
 - Konstruksi respons
3. Jika alat gagal atau mengembalikan data yang tidak mencukupi, agen dapat merencanakan ulang atau menyusun ulang tugas secara dinamis.
4. Memori (misalnya, penyimpanan vektor jangka pendek) dapat mempertahankan statusnya di seluruh langkah

Takeaways

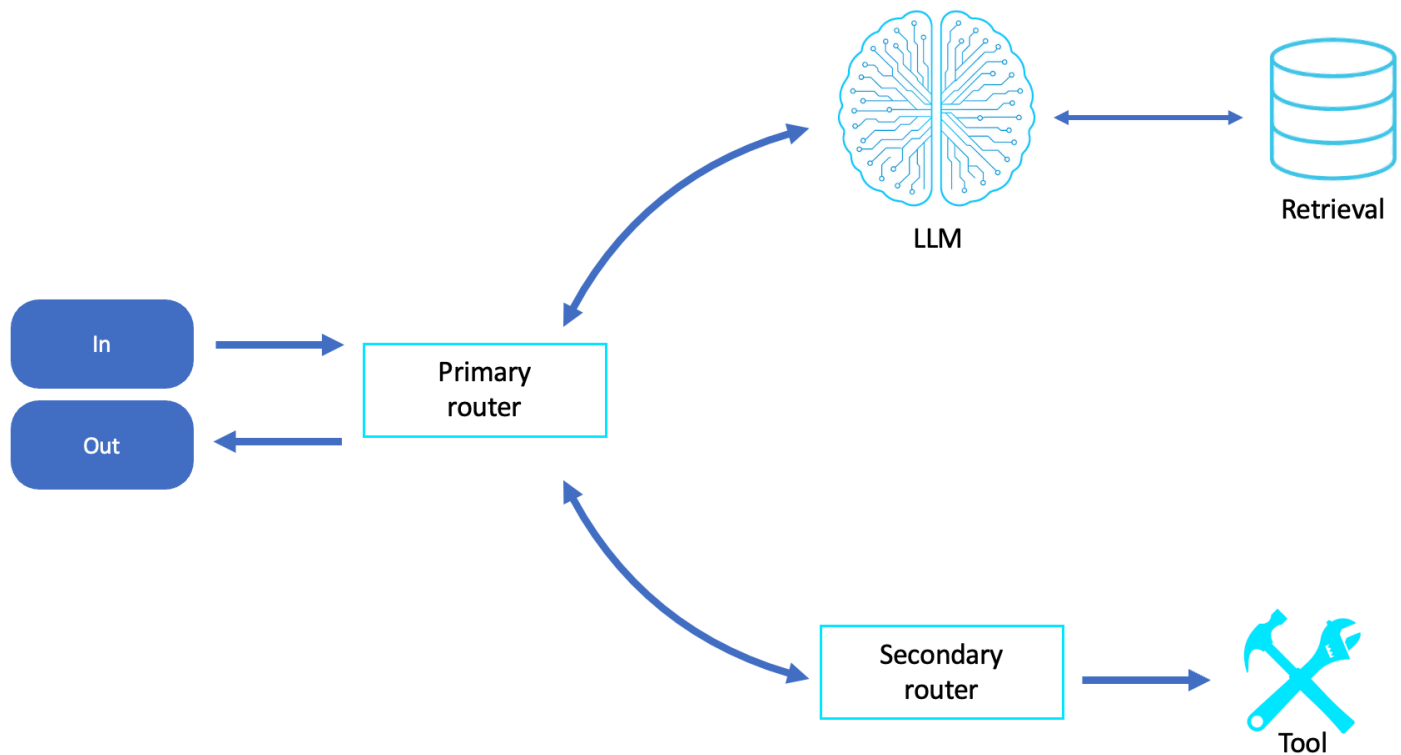
Di mana pola saga mengelola panggilan layanan terdistribusi dengan logika kompensasi, rantai cepat mengelola tugas penalaran dengan pengurutan reflektif dan perencanaan ulang adaptif. Kedua sistem memungkinkan kemajuan tambahan, poin keputusan terdesentralisasi, dan pemulihan kegagalan, dan melakukan semua ini melalui penalaran yang diinformasikan daripada kemunduran yang kaku.

Prompt chaining memperkenalkan penalaran transaksional, yang merupakan padanan kognitif dari saga. Artinya, setiap “pemikiran” dievaluasi kembali, direvisi, atau ditinggalkan sebagai bagian dari dialog yang diarahkan pada tujuan yang lebih luas.

Merutekan pola pengiriman dinamis

Dalam sistem agen modern, di mana tugas berkisar dari penguraian dokumen hingga pembuatan perangkat lunak otonom, kemampuan untuk merutekan permintaan secara dinamis ke model bahasa besar (LLM) atau agen yang paling mampu menjadi penting. Logika perutean statis, sering tertanam dalam skrip orkestrasi atau lapisan API, tidak memiliki kemampuan beradaptasi yang diperlukan untuk lingkungan real-time, multi-model, multi-kemampuan. Untuk mengatasi hal ini, alur kerja perutean LLM dapat diubah menjadi arsitektur berbasis peristiwa yang memanfaatkan pola pengiriman dinamis, mengubah panggilan LLM menjadi peristiwa yang diarahkan secara cerdas dan sadar konteks.

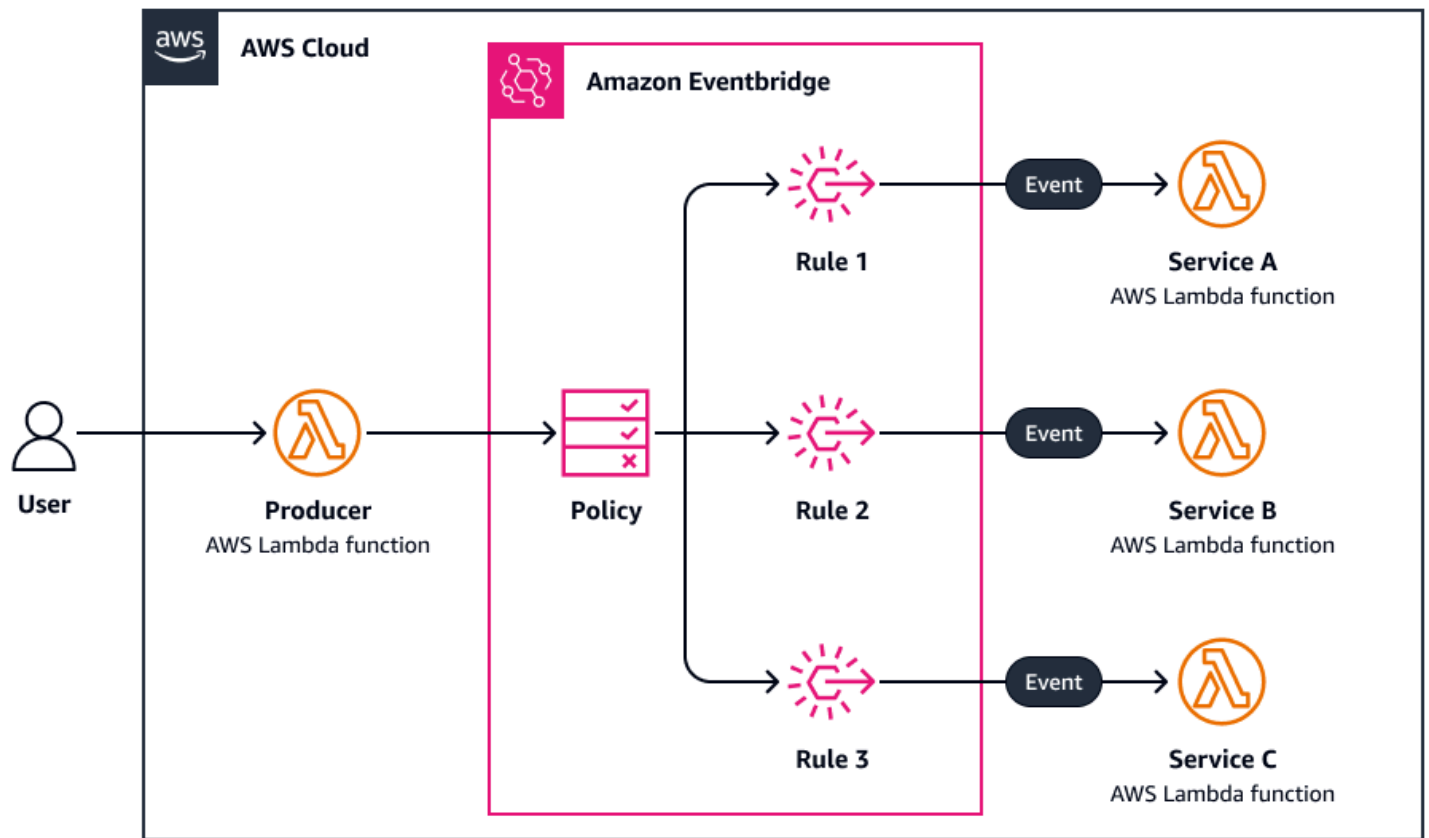
Diagram berikut adalah contoh perutean LLM:



Pengiriman dinamis

Dalam sistem terdistribusi tradisional, pola pengiriman dinamis memilih dan memanggil layanan tertentu saat runtime berdasarkan atribut peristiwa yang masuk, seperti jenis peristiwa, sumber, dan muatan. Ini biasanya diterapkan menggunakan Amazon EventBridge, yang dapat mengevaluasi dan merutekan peristiwa yang masuk ke target yang sesuai (misalnya, AWS Lambda fungsi AWS Step Functions, atau tugas Amazon Elastic Container Service).

Diagram berikut adalah contoh pengiriman dinamis:



1. Aplikasi memancarkan peristiwa (misalnya, `{"type": "orderCreated", "priority": "high"}`).
2. Amazon EventBridge mengevaluasi acara tersebut terhadap aturan peruteannya.
3. Berdasarkan atribut acara, sistem secara dinamis mengirimkan ke yang berikut:
 - `HighPriorityOrderProcessor`(layanan A)
 - `StandardOrderProcessor`(layanan B)
 - `UpdateOrderProcessor`(layanan C)

Pola ini mendukung kopling longgar, spesialisasi berbasis domain, dan ekstensibilitas runtime. Hal ini memungkinkan sistem untuk merespons secara cerdas terhadap perubahan persyaratan dan semantik peristiwa.

LLM-based perutean

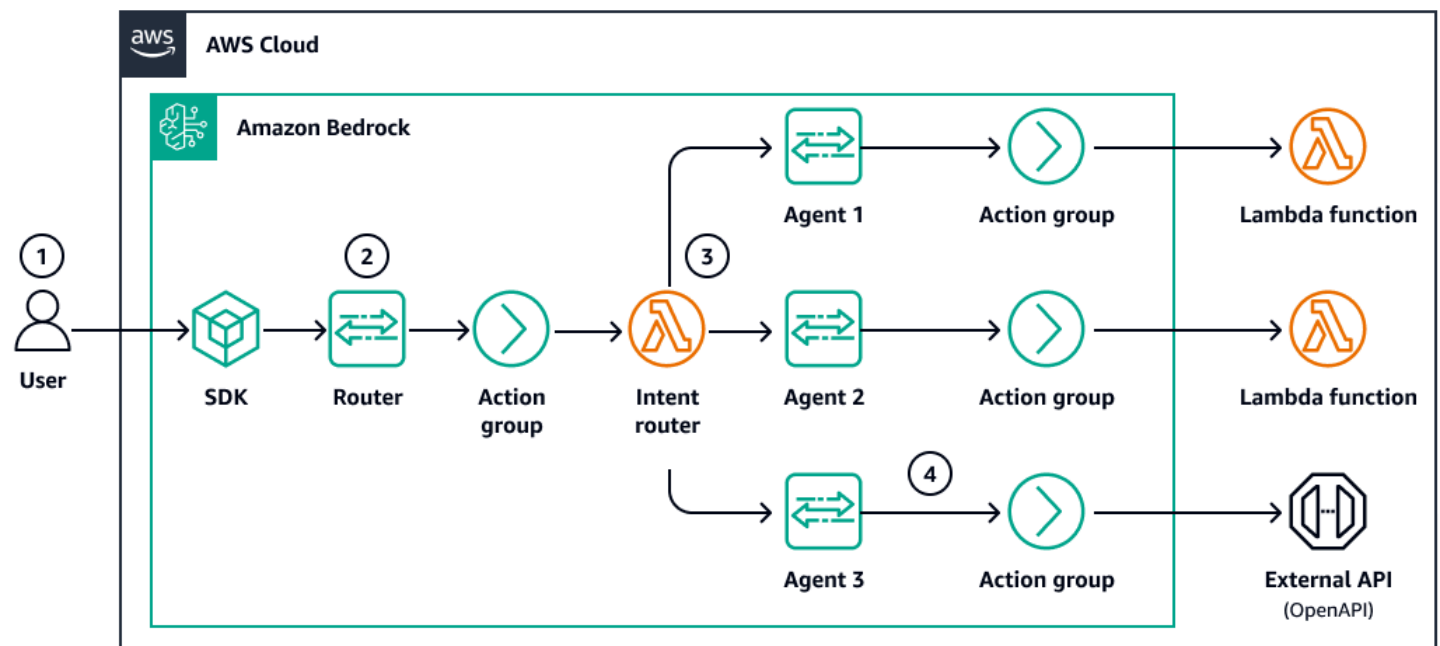
Dalam sistem agen, perutean juga melakukan pendelegasian tugas dinamis — tetapi alih-alih EventBridge aturan Amazon atau filter metadata, LLM mengklasifikasikan dan menafsirkan maksud pengguna melalui bahasa alami. Hasilnya adalah bentuk pengiriman yang fleksibel, semantik, dan adaptif.

Router agen

Arsitektur ini memungkinkan pengiriman berbasis niat yang kaya tanpa skema atau jenis acara yang telah ditentukan sebelumnya, yang ideal untuk input tidak terstruktur dan kueri kompleks.

1. Seorang pengguna mengajukan permintaan “Dapatkan Anda membantu saya meninjau persyaratan kontrak saya?”
2. LLM menafsirkan ini sebagai tugas dokumen hukum.
3. Agen merutekan tugas ke satu atau beberapa hal berikut:
 - Templat prompt peninjauan kontrak
 - Penalaran hukum subagent
 - Alat penguraian dokumen

Diagram berikut adalah contoh router agen:



1. Pengguna mengirimkan permintaan bahasa alami melalui SDK.
2. Agen Amazon Bedrock menggunakan LLM untuk mengklasifikasikan tugas (misalnya, hukum, teknis, atau penjadwalan).
3. Agen secara dinamis merutekan tugas melalui grup tindakan untuk memanggil agen yang diperlukan:
 - Domain-specific agen

- Rantai alat khusus
- Konfigurasi prompt kustom

4. Handler yang dipilih memproses tugas dan mengembalikan respons yang disesuaikan.

Takeaways

Jika pengiriman dinamis tradisional menggunakan EventBridge aturan Amazon untuk perutean berdasarkan atribut peristiwa terstruktur, perutean agen menggunakan LLM untuk secara semantik mengklasifikasikan dan merutekan tugas berdasarkan makna dan maksud. Ini memperluas fleksibilitas sistem dengan mengaktifkan yang berikut:

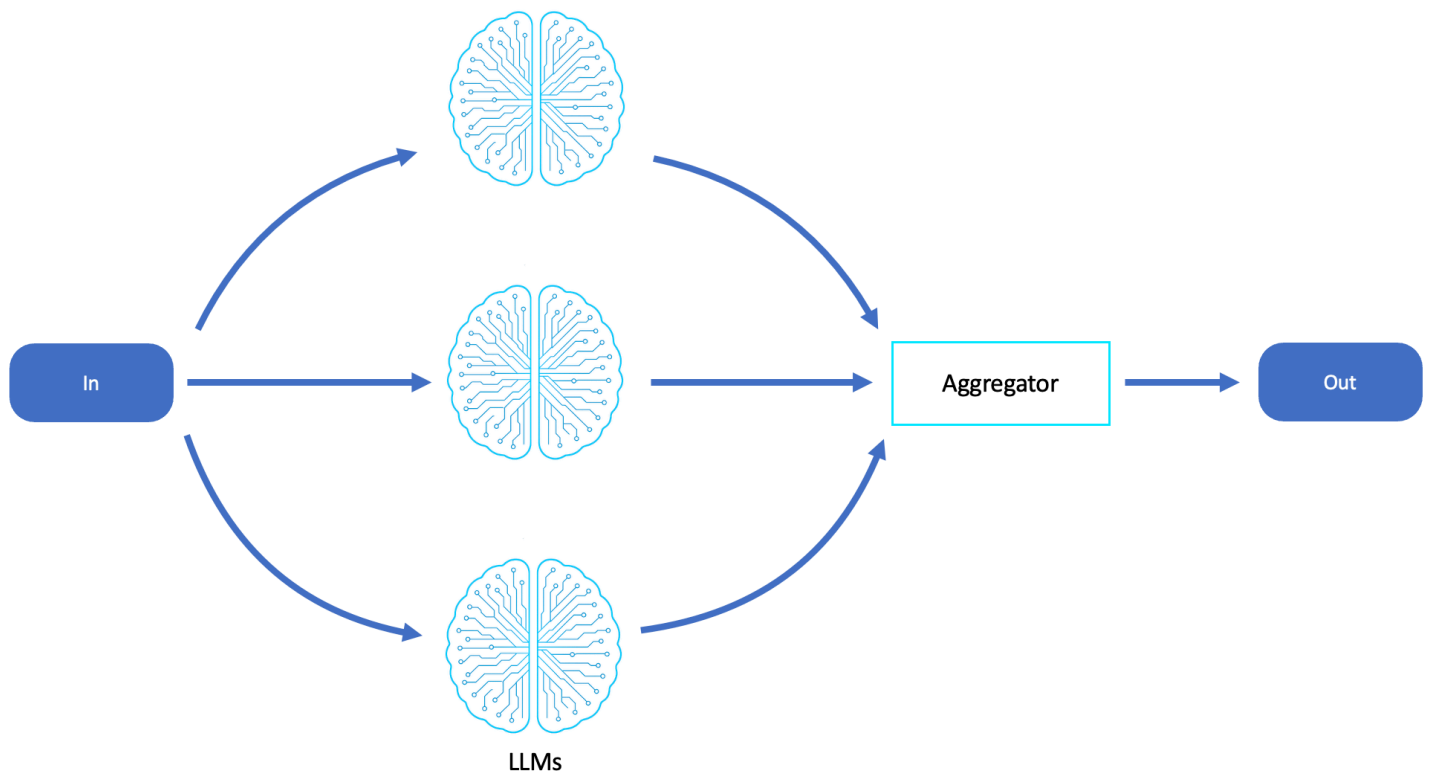
- Pemahaman masukan yang lebih luas
- Fallback cerdas dan pemilihan alat
- Ekstensibilitas alami melalui peran agen baru atau gaya prompt

Perutean agen menggantikan aturan kaku dengan pengiriman kognitif dinamis, yang memungkinkan sistem berkembang dengan bahasa daripada kode.

Paralelisasi dan pola scatter-gathering

Banyak tugas penalaran dan pembuatan tingkat lanjut - seperti meringkas dokumen besar, mengevaluasi beberapa jalur solusi, atau membandingkan perspektif yang beragam - mendapat manfaat dari eksekusi paralel petunjuk. Alur kerja berurutan tradisional gagal ketika skalabilitas, responsif, dan toleransi kesalahan diperlukan. Untuk mengatasinya, LLM-based paralelisasi dapat ditata ulang menggunakan pola scatter-gathering yang digerakkan oleh peristiwa, di mana tugas secara dinamis menyebar ke agen otonom dan hasilnya disintesis secara cerdas.

Diagram berikut adalah contoh alur kerja paralelisasi LLM:



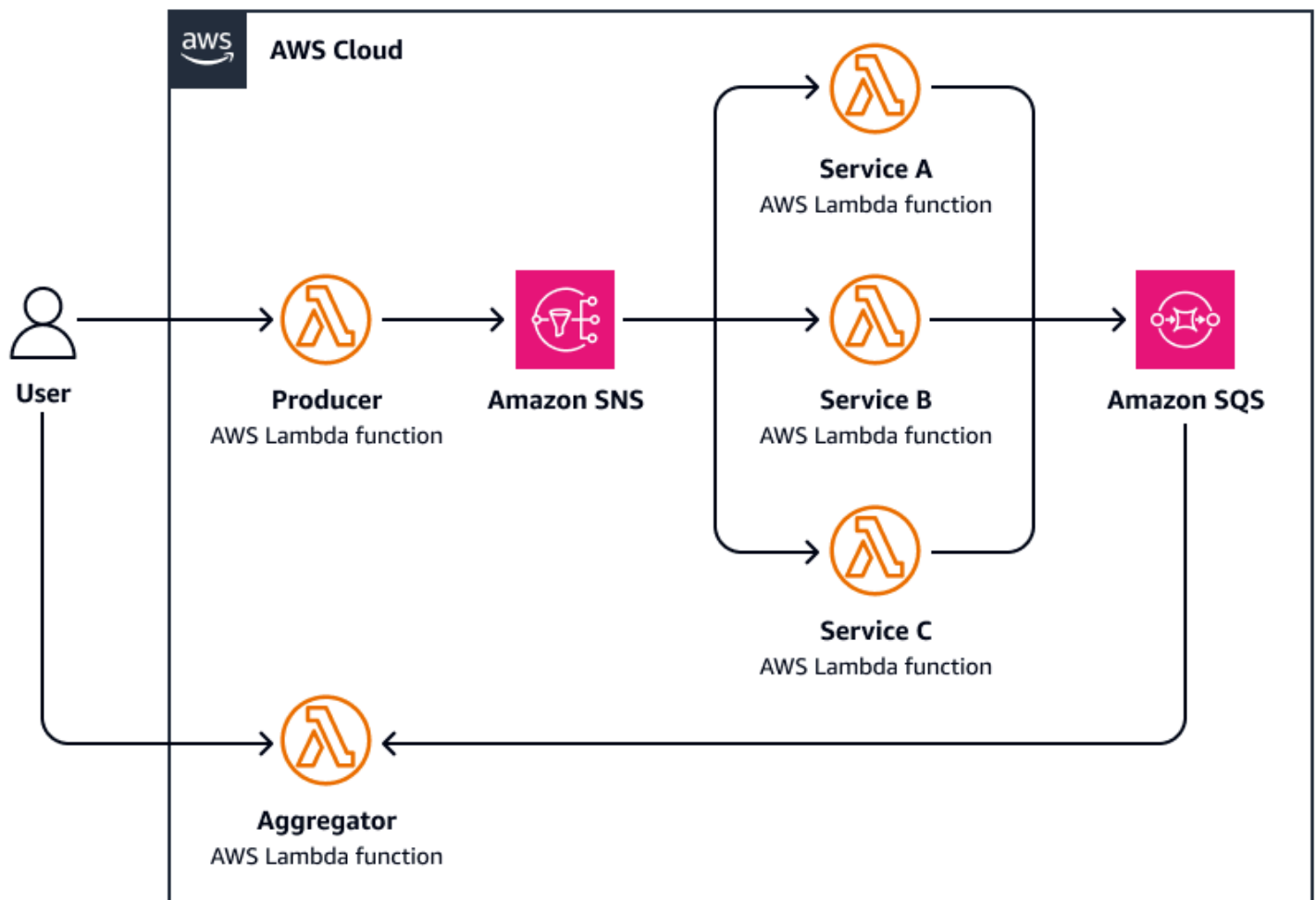
Scatter-gather

Dalam sistem terdistribusi, pola scatter-gathering mengirimkan tugas ke beberapa layanan atau unit pemrosesan secara paralel, menunggu tanggapan mereka, dan kemudian mengumpulkan hasil menjadi output terkonsolidasi. Tidak seperti fan-out, scatter-gathering dikoordinasikan karena mengharapkan respons dan biasanya menerapkan logika untuk menggabungkan, membandingkan, dan memilih hasil.

Implementasi umum untuk paralelisasi dan scatter-gathering meliputi:

- AWS Step Functions memetakan status untuk eksekusi tugas paralel
- AWS Lambda dengan konkurensi, mengoordinasikan hasil dari beberapa fungsi yang dipanggil
- Amazon EventBridge dengan ID korelasi dan alur kerja agregasi
- Pola pengontrol khusus untuk mengelola fan-out dan mengumpulkan hasil dengan menggunakan Amazon Simple Storage Service (Amazon S3), Amazon DynamoDB, atau antrian

Diagram berikut adalah contoh scatter-gathering:



1. Pengguna mengirimkan permintaan ke fungsi koordinator pusat yang menyebarkan tugas dengan menerbitkan pesan paralel ke topik Amazon Simple Notification Service (Amazon SNS).
2. Setiap pesan mencakup metadata tugas dan diarahkan ke pekerja khusus. AWS Lambda
3. Setiap pekerja AWS Lambda secara independen memproses subtugas yang ditetapkan (misalnya, menanyakan API eksternal, memproses dokumen, dan menganalisis data).
4. Hasil ditulis ke lapisan penyimpanan umum, seperti Amazon Simple Queue Service (Amazon SQS).
5. Fungsi agregator menunggu semua tanggapan selesai, dan kemudian melakukan hal berikut:
 - Mengumpulkan dan mengumpulkan hasil (misalnya, menggabungkan ringkasan, memilih kecocokan terbaik)
 - Mengirim respons akhir atau memicu alur kerja hilir

Kasus penggunaan umum untuk pola scatter-gathering meliputi yang berikut:

- Pencarian federasi
- Mesin perbandingan harga
- Analisis data agregat
- Inferensi multimodel

LLM-based paralelisasi (kognisi sebar-kumpulkan)

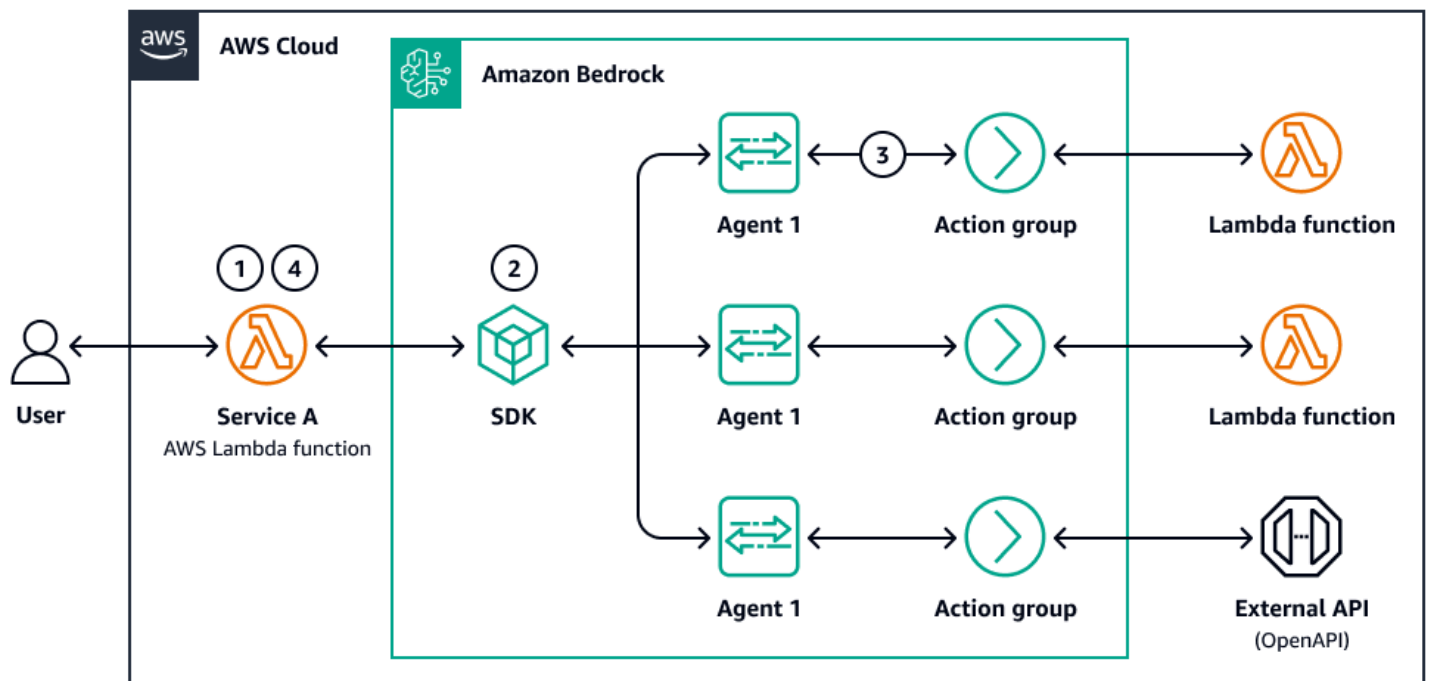
Dalam sistem agen, paralelisasi sangat mencerminkan scatter-gathering dengan mendistribusikan subtugas di beberapa panggilan atau agen LLM, masing-masing secara independen bernalar melalui sebagian masalah. Hasil yang dikembalikan dikumpulkan dan disintesis oleh proses agregasi, yang seringkali merupakan LLM atau agen pengontrol lainnya.

Paralelisasi agen

1. Seorang agen mengajukan permintaan “Ringkas wawasan di 10 laporan ini.”
2. Ini menyebarkan laporan ke 10 tugas ringkasan LLM paralel.
3. Ketika mengembalikan semua ringkasan, agen melakukan hal berikut:
 - Agregat ringkasan menjadi briefing terpadu
 - Mengidentifikasi tema atau kontradiksi
 - Mengirim output yang disintesis ke pengguna

Alur kerja agen ini memungkinkan penalaran paralel yang dapat diskalakan, modular, dan adaptif. Ini sangat ideal untuk kasus penggunaan yang membutuhkan throughput kognitif tinggi.

Diagram berikut adalah contoh paralelisasi agen:



1. Seorang pengguna mengirimkan kueri multipart atau set dokumen.
2. Pengontrol AWS Lambda atau fungsi langkah mendistribusikan subtugas. Setiap tugas memanggil panggilan Amazon Bedrock LLM atau subagent dengan prompt sendiri.
3. Ketika panggilan dan subtugas selesai, hasil disimpan (misalnya, di Amazon S3 atau penyimpanan memori), dan langkah agregasi menggabungkan, membandingkan, atau memfilter output.
4. Sistem mengembalikan respons akhir kepada pengguna atau agen hilir.

Sistem ini memiliki loop penalaran terdistribusi dengan ketertelusuran, toleransi kesalahan, dan pembobotan hasil opsional atau logika seleksi.

Takeaways

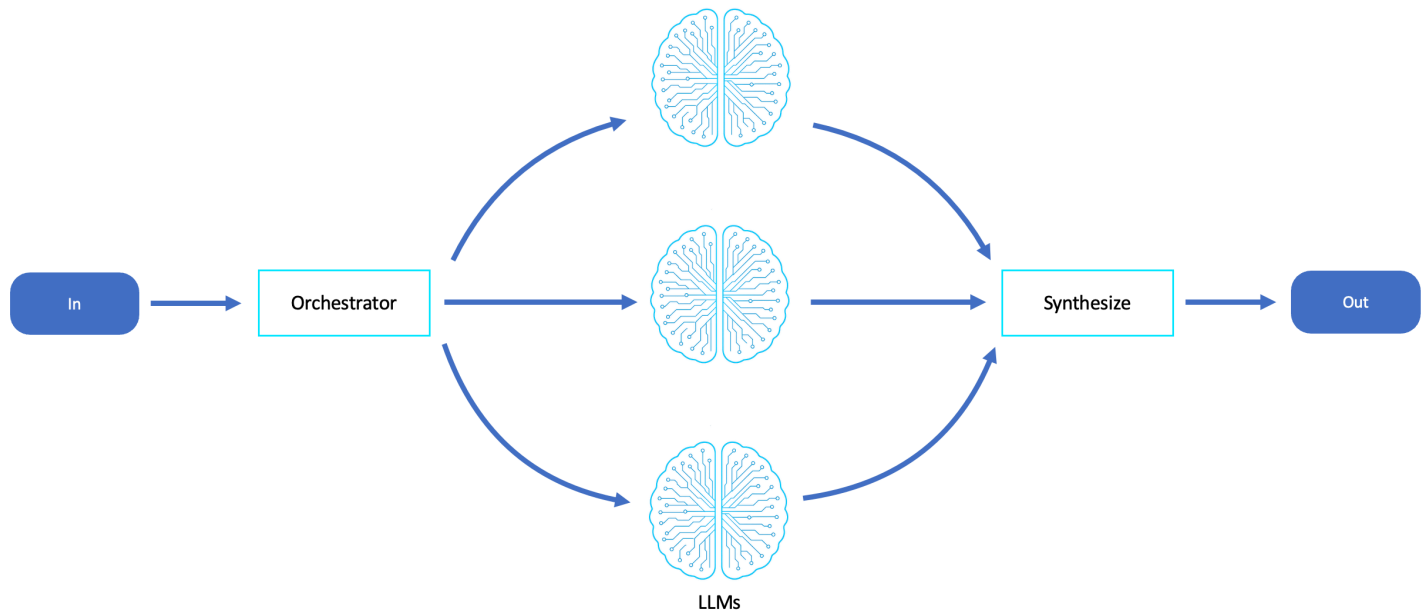
Paralelisasi agen menggunakan pola scatter-gathering untuk mendistribusikan tugas LLM, memungkinkan pemrosesan paralel dan sintesis hasil cerdas.

Pola orkestrasi Saga

Karena alur kerja yang didorong oleh LLM menjadi semakin kompleks, mencakup rantai cepat, langkah-langkah pemrosesan data, pemanggilan alat, dan kolaborasi agen, kebutuhan akan orkestrasi cerdas menjadi penting. Daripada mengandalkan skrip yang digabungkan secara ketat

atau alur eksekusi statis yang telah ditentukan sebelumnya, alur kerja ini dapat diimplementasikan sebagai pola orkestrasi yang digerakkan oleh peristiwa, memungkinkan LLM-based sistem untuk secara dinamis mengoordinasikan, memantau, dan mengadaptasi tugas multi-langkah di seluruh agen otonom.

Diagram berikut adalah contoh orkestrator:



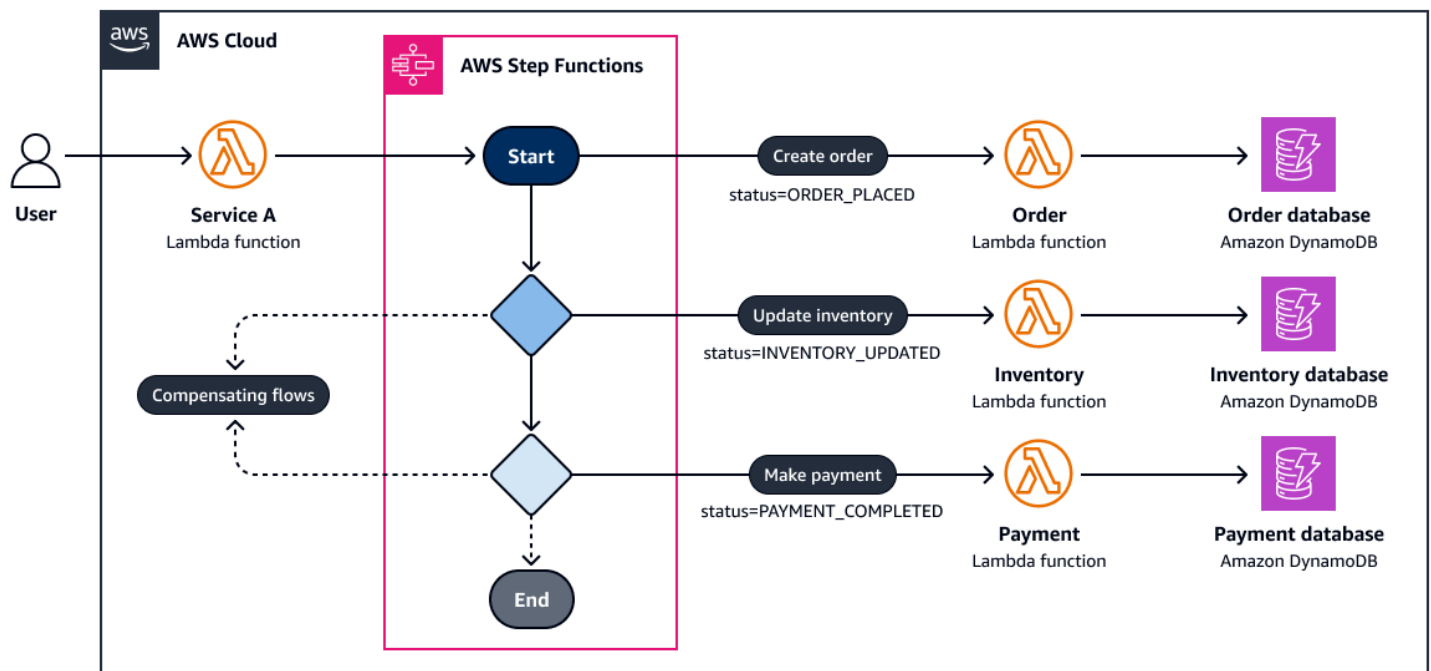
Orkestrasi acara

Dalam sistem terdistribusi tradisional, orkestrasi peristiwa mengacu pada pola di mana koordinator pusat mengelola alur kerja yang kompleks dengan secara eksplisit mengarahkan aliran kontrol di berbagai layanan atau tugas. Tidak seperti koreografi acara (di mana setiap layanan bereaksi secara independen), orkestrasi memberikan logika terpusat, visibilitas, dan kontrol atas seluruh proses.

Ini biasanya diimplementasikan dengan menggunakan alat-alat berikut:

- AWS Step Functions— Tentukan dan jalankan alur kerja stateful
- AWS Lambda— Melaksanakan tugas-tugas diskrit dalam aliran yang diatur
- Amazon SQS atau Amazon EventBridge - Memicu langkah atau respons asinkron

Diagram berikut adalah contoh orkestrasi saga:



AWS Step Functions Alur kerja mengelola proses pesanan pelanggan:

1. Buat pesanan (AWS Lambda)
2. Perbarui inventaris (AWS Lambda)
3. Lakukan pembayaran (AWS Lambda)

Orkestrator mengoordinasikan setiap langkah dengan mengelola percobaan ulang, cabang paralel, batas waktu, dan kegagalan.

Role-based sistem agen (orkestrator)

Dalam sistem agen, pola orkestrator mencerminkan orkestrasi peristiwa tetapi mendistribusikan logika di beberapa agen penalaran, masing-masing dengan peran atau spesialisasi yang ditentukan. Agen orkestrator pusat menafsirkan tugas keseluruhan, menguraikannya menjadi subtask, dan mendelegasikannya ke agen pekerja, masing-masing dioptimalkan untuk domain tertentu (misalnya, penelitian, pengkodean, ringkasan, tinjauan).

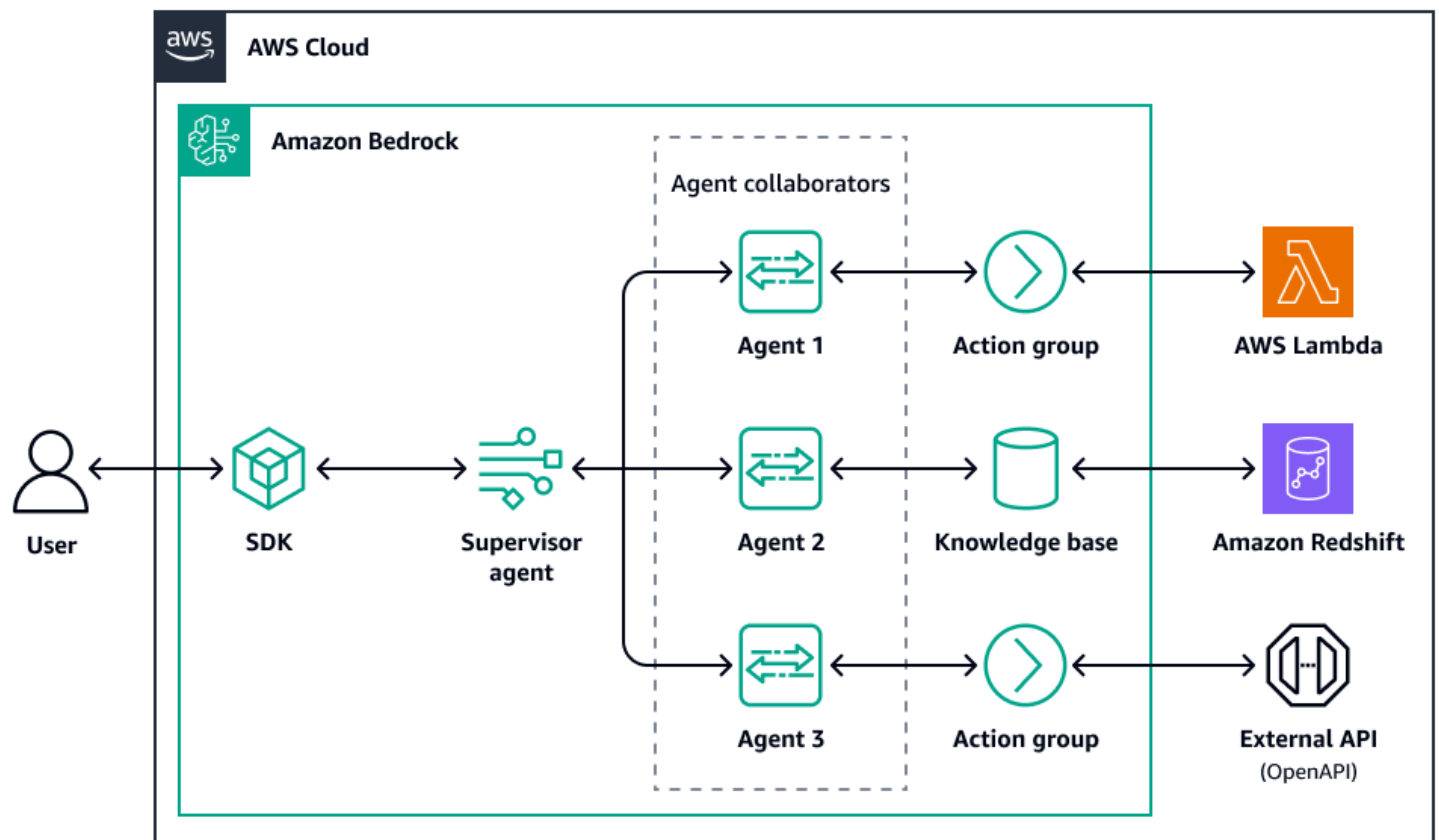
Pengawas

1. Seorang pengguna mengirimkan kueri "Buat ringkasan proyek dan rangkum 5 pesaing teratas."
2. Agen orkestrator melakukan hal berikut:

- Menugaskan agen riset untuk menemukan data pesaing
- Mengirim temuan mentah ke agen ringkasan
- Memberikan hasil ke agen penulis singkat
- Mengkompilasi output akhir untuk pengguna

Setiap agen beroperasi secara independen, tetapi orkestrator mengoordinasikan tugas. Ini seperti fungsi Lambda yang menangani tugas alur kerja.

Diagram berikut contoh seorang supervisor:



1. Seorang pengguna mengirimkan tugas ke agen supervisor Amazon Bedrock.
2. Agen supervisor mem-parsing permintaan menjadi subtugas untuk setiap kolaborator agen.
3. Setiap subtugas ditugaskan ke agen kolaborator dengan prompt atau rantai alat khusus peran.
4. Agen pekerja memanggil API atau alat eksternal melalui grup tindakan.
5. Setiap agen pekerja mengembalikan output dalam format terstruktur.
6. Ketika semua pekerja mengembalikan hasil mereka, supervisor mengevaluasi, mensintesis, dan mengembalikan respons akhir.

Struktur ini memungkinkan modularitas, kemampuan beradaptasi, dan introspeksi di seluruh alur kerja agen multistep yang kompleks.

Takeaways

Di mana orkestrasi peristiwa menggunakan kontrol terpusat (misalnya, AWS Step Functions) untuk mengarahkan eksekusi layanan, sistem agen berbasis peran menggunakan agen LLM-powered orkestrator untuk bernalar tentang tujuan, mendelegasikan subtugas ke agen pekerja, dan mensintesis hasil akhir.

Dalam kedua paradigma, orkestrator melakukan hal berikut:

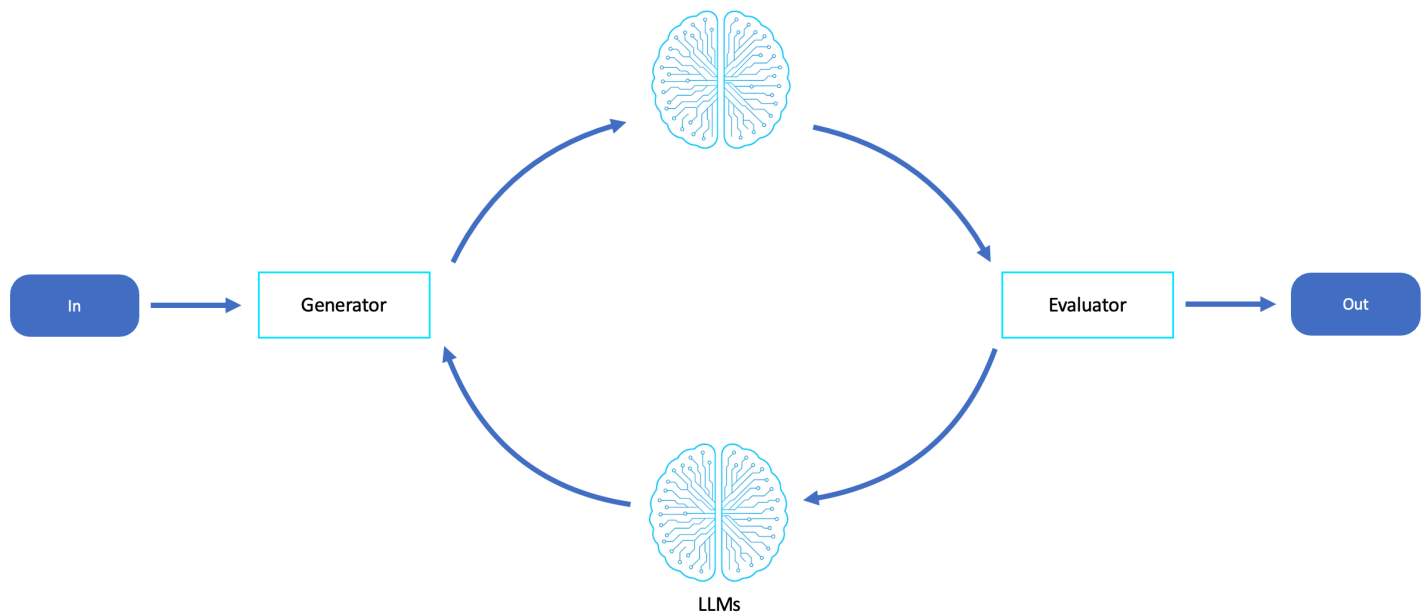
- Mempertahankan konteks dan alur eksekusi
- Menangani percabangan, pengurutan, dan penanganan kesalahan
- Menghasilkan hasil terpadu dari komponen terdistribusi

Orkestrasi agen, bagaimanapun, menambah penalaran, kemampuan beradaptasi, dan delegasi semantik. Ini membuatnya sangat cocok untuk tugas terbuka, ambigu, dan berkembang.

Evaluasi refleksi-memperbaiki pola loop

Tugas-tugas seperti pembuatan kode, ringkasan, atau pengambilan keputusan otonom mendapat manfaat besar dari umpan balik runtime, memungkinkan sistem untuk berkembang melalui pengamatan dan penyempurnaan. Untuk mengoperasionalkan ini, siklus reflekt-perbaiki dapat diimplementasikan sebagai loop kontrol umpan balik yang digerakkan oleh peristiwa - pola yang terinspirasi oleh rekayasa sistem, yang disesuaikan untuk alur kerja yang otonom dan cerdas.

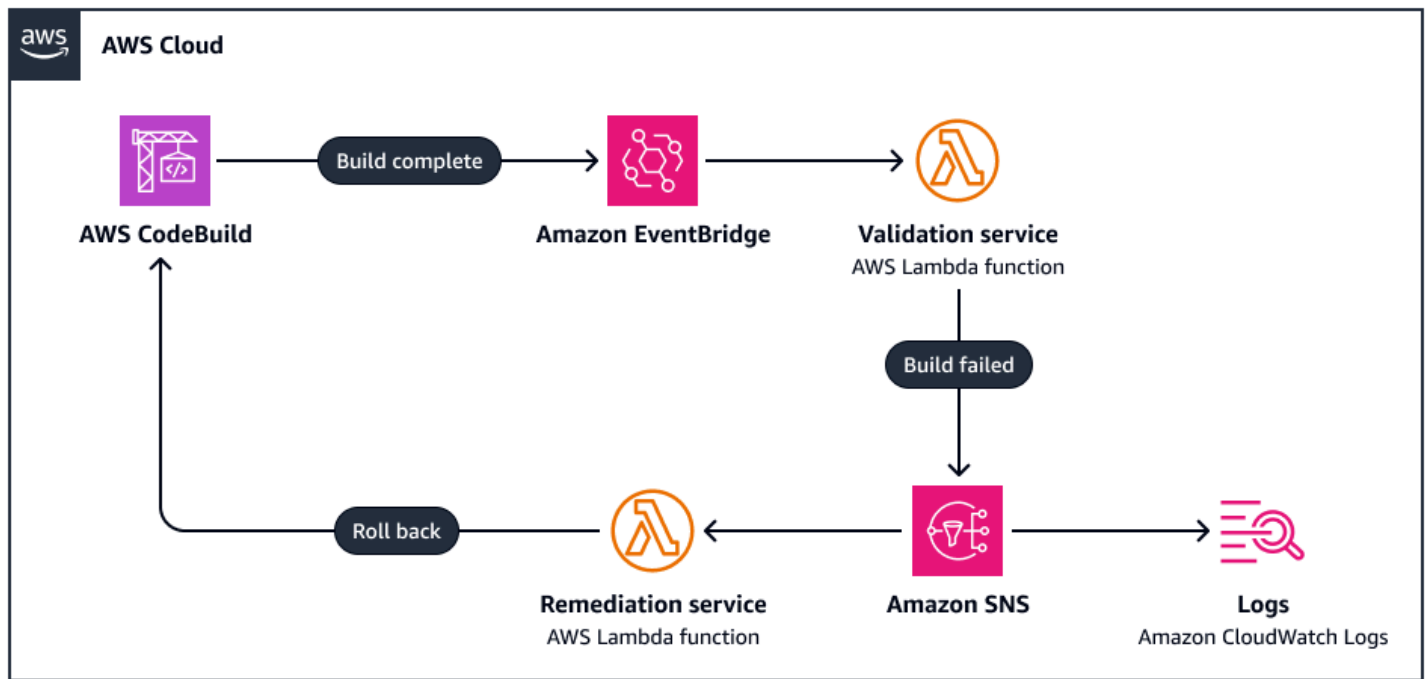
Diagram berikut adalah contoh dari evaluator reflect-refine feedback loop:



Lingkaran kontrol umpan balik

Lingkaran kontrol umpan balik adalah pola yang memantau keluaran dan perilakunya sendiri, mengevaluasinya terhadap kriteria yang ditentukan atau keadaan yang diinginkan, dan kemudian menyesuaikan tindakannya sesuai dengan itu. Arsitektur ini terinspirasi oleh teori kontrol dan merupakan dasar dalam domain seperti otomatisasi, integrasi berkelanjutan dan jalur pipa pengiriman berkelanjutan (CI/CD), dan operasi pembelajaran mesin.

Diagram berikut adalah contoh loop kontrol umpan balik:



1. Pipeline penerapan memancarkan peristiwa BuildComplete.
2. Peristiwa ini memicu pekerjaan pengujian atau evaluasi otomatis yang memvalidasi build.
3. Jika validasi gagal (misalnya, karena pengujian yang gagal, masalah keamanan, atau pelanggaran kebijakan), sistem:
 - Memancarkan acara BuildComplete
 - Mencatat masalah atau mengirim pemberitahuan
 - Memicu remediasi atau tindakan korektif, seperti rollback, patching, atau coba lagi

Loop berlanjut sampai menghasilkan hasil atau eskalasi yang dapat diterima, atau time out terjadi. Pola ini biasanya digunakan untuk hal-hal berikut:

- EventBridge Aturan Amazon untuk merutekan acara ke tugas evaluasi atau remediasi
- AWS Step Functions untuk logika coba lagi berulang dan percabangan pada hasil evaluasi
- Amazon Simple Notification Service (Amazon SNS) atau alarm CloudWatch Amazon untuk pemicu dan peringatan umpan balik
- AWS Lambda fungsi atau pekerja kontainer untuk menerapkan tindakan korektif

Lingkaran kontrol umpan balik (evaluator)

Alur kerja evaluator adalah loop umpan balik kognitif yang didukung oleh LLM atau agen penalaran. Prosesnya terdiri dari yang berikut:

1. Agen generator atau LLM menghasilkan output (misalnya, rencana, jawaban, atau draf).
2. Agen evaluator meninjau hasilnya menggunakan prompt kritik atau rubrik evaluasi.
3. Berdasarkan umpan balik, agen asli atau agen pengoptimal baru merevisi output.

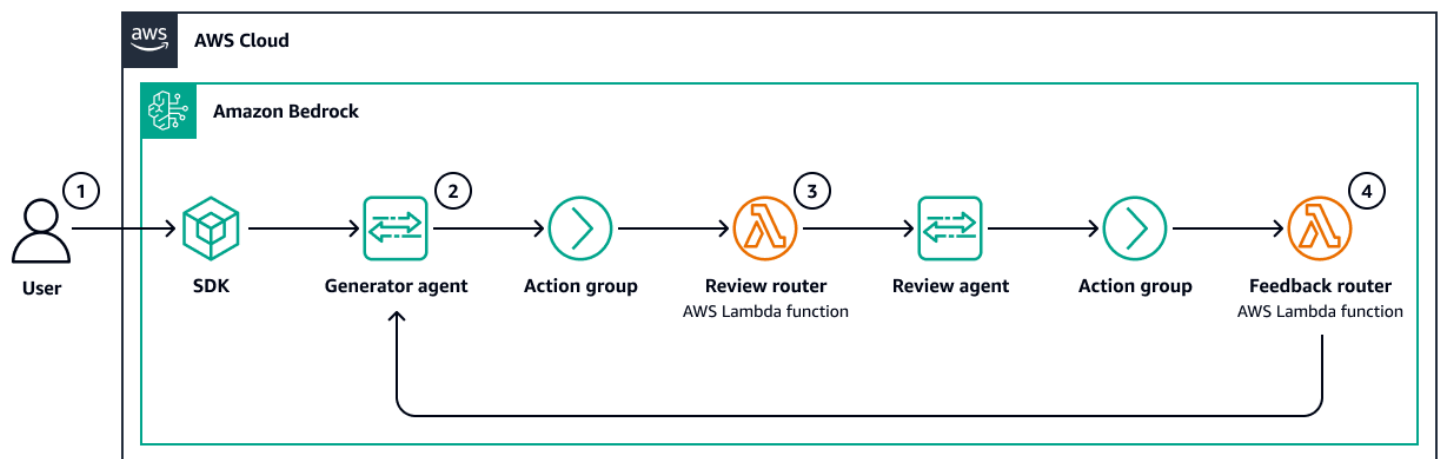
Loop berulang sampai hasilnya memenuhi serangkaian kriteria, disetujui, atau mencapai batas coba lagi.

Penilai

1. Seorang pengguna meminta agen untuk menulis ringkasan kebijakan.
2. Agen generator menyusunnya.
3. Agen evaluator memeriksa cakupan, nada, dan kebenaran hukum.
4. Jika respons tidak memadai, itu disempurnakan dan dikirim ulang hingga loop umpan balik menyatu.

Ini memungkinkan penilaian diri, penyempurnaan berulang, dan kontrol keluaran adaptif — semuanya tanpa masukan manusia.

Diagram berikut adalah contoh loop kontrol umpan balik (evaluator):



1. Seorang pengguna mengeluarkan tugas (misalnya, menyusun strategi bisnis).

2. Agen Amazon Bedrock menghasilkan draf awal menggunakan LLM.
3. Agen kedua (atau prompt tindak lanjut) melakukan evaluasi terstruktur (misalnya, “nilai output ini dengan kejelasan, kelengkapan, dan nada”).
4. Jika peringkat jatuh di bawah ambang batas, respons direvisi oleh:
 - Menerapkan kembali generator dengan kritik tertanam
 - Mengirim umpan balik ke agen penyuling khusus
 - Iterasi sampai respon yang dapat diterima tercapai

Komponen opsional seperti AWS Lambda pengontrol atau AWS Step Functions dapat mengelola ambang umpan balik, percobaan ulang, dan strategi mundur.

Takeaways

Di mana loop kontrol umpan balik tradisional menggunakan peristiwa, metrik, dan logika remediasi untuk memvalidasi dan menyesuaikan perilaku sistem, loop evaluator agen menggunakan agen penalaran untuk mengevaluasi, merefleksikan, dan merevisi output secara dinamis.

Dalam kedua paradigma:

- Output dievaluasi setelah dihasilkan
- Tindakan korektif atau pemurnian dipicu berdasarkan umpan balik
- Sistem terus beradaptasi terhadap kualitas target atau tujuan

Versi agen mengubah validasi statis menjadi refleksi semantik, memungkinkan agen peningkatan diri yang mengevaluasi efektivitas mereka sendiri.

Merancang alur kerja agen pada AWS

Setiap pola dalam panduan ini dapat dibangun menggunakan Layanan AWS. Agen Amazon Bedrock menyediakan orkestrasi, akses data, dan saluran interaksi.

Komponen	Layanan AWS	Tujuan
Alasan LLM	Amazon Bedrock	Logika agen, perencanaan, penggunaan alat

Eksekusi alat	AWS Lambda, Amazon ECS, Amazon SageMaker	Host alat eksternal untuk agen
Memori dan RAG	Basis pengetahuan Amazon Bedrock, Amazon S3, OpenSearch	Memori persisten dan semantik
Orkestrasi	AWS Step Functions	Tugas multistep dan koordinasi agen
Perutean acara	Amazon EventBridge, Amazon SQS	Pesan interagen terpisah
Antarmuka pengguna	Amazon API Gateway, AWS AppSync, SDK	Titik masuk untuk aplikasi atau sistem
Memantau	Amazon CloudWatch, AWS X-Ray, AWS Distro untuk OpenTelemetry	Observabilitas dan introspeksi agen

Kesimpulan

Pola alur kerja agen adalah tahap evolusi berikutnya dari arsitektur berbasis peristiwa, di mana logika bisnis tidak didefinisikan secara statis tetapi beralasan secara dinamis melalui penggunaan kognisi yang ditingkatkan model bahasa besar (LLM). Dengan menggabungkan primitif cloud-native tradisional dengan alur kerja LLM dan pola desain agen, organisasi dapat membangun sistem adaptif, cerdas, dan modular yang merespons dengan tujuan dan belajar dari pengalaman.

Dalam pola ini, Amazon Bedrock adalah pintu gerbang menuju kognisi agen, memungkinkan LLM-based agen mengakses alur kerja peristiwa, berinteraksi dengan alat dan memori, dan memberikan hasil yang terstruktur, dapat dilacak, dan selaras.

Saat Anda merancang dan menerapkan sistem agen, pola alur kerja ini menyediakan cetak biru untuk membangun arsitektur AI yang otonom dan dapat dikomposisi. Sistem ini didasarkan pada praktik terbaik tanpa server dan ditambah dengan model pondasi cerdas.

Riwayat dokumen

Tabel berikut menjelaskan perubahan signifikan pada panduan ini. Jika Anda ingin diberi tahu tentang pembaruan masa depan, Anda dapat berlangganan umpan [RSS](#).

Perubahan	Deskripsi	Tanggal
Publikasi awal	—	Juli 14, 2025

Terjemahan disediakan oleh mesin penerjemah. Jika konten terjemahan yang diberikan bertentangan dengan versi bahasa Inggris aslinya, utamakan versi bahasa Inggris.