Deploying multi-agent system in production at scale

The better the question. The better the answer. The better the world works.

EY

Shape the future with confidence

Prototypes are easy. Production is, well, hard!

Many organizations embrace agentic AI to automate tasks and boost efficiency. But does a working multi-agent workflow mean we have a system ready for real-world production?

What is multi-agent AI systems?

An AI multi-agent system is a distributed system composed of multiple intelligent agents that can sense, learn, and act autonomously to achieve individual and collective goals.

Multi agent capabilities:

- Flexibility and scalability
- Robustness and reliability
- Self-organization and coordination
- Real-time operation

Project manager agent

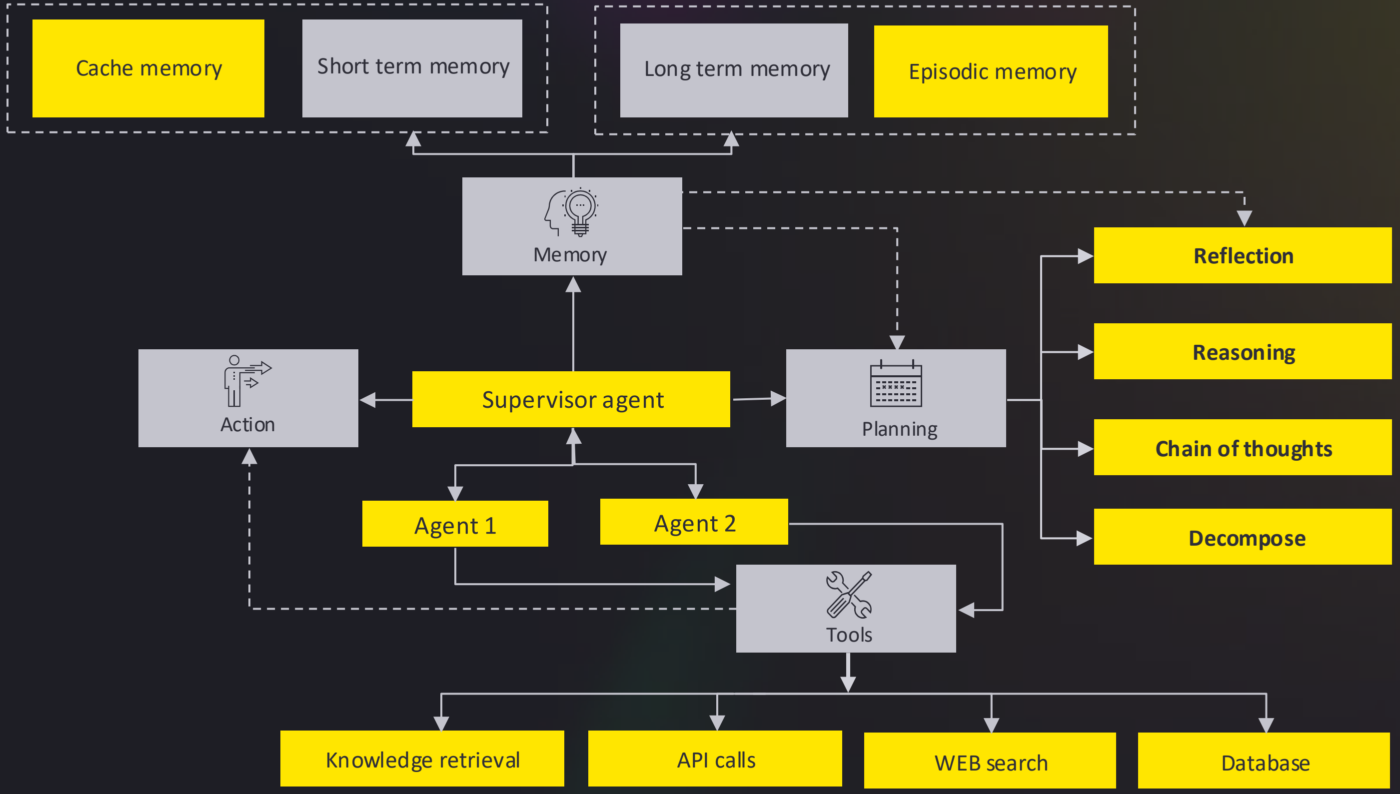
- Task: Oversees project management
- Process: Coordinating efforts among agents and ensuring the process aligns with overall project goals
- Output: Smoothly running project that stays on target

Content developer agent

- Task: Creates content, writes stuff, and helps make documents
- Process: Creates ideas, writes them down, and polishes them
- Output: Written content and materials for the project

Market research analyst agent

- Task: Looks for market info, studies trends, and gives advice based on what they find
- Process: Collects data, analyses it, and draws conclusions
- Output: Reports and insights about the market



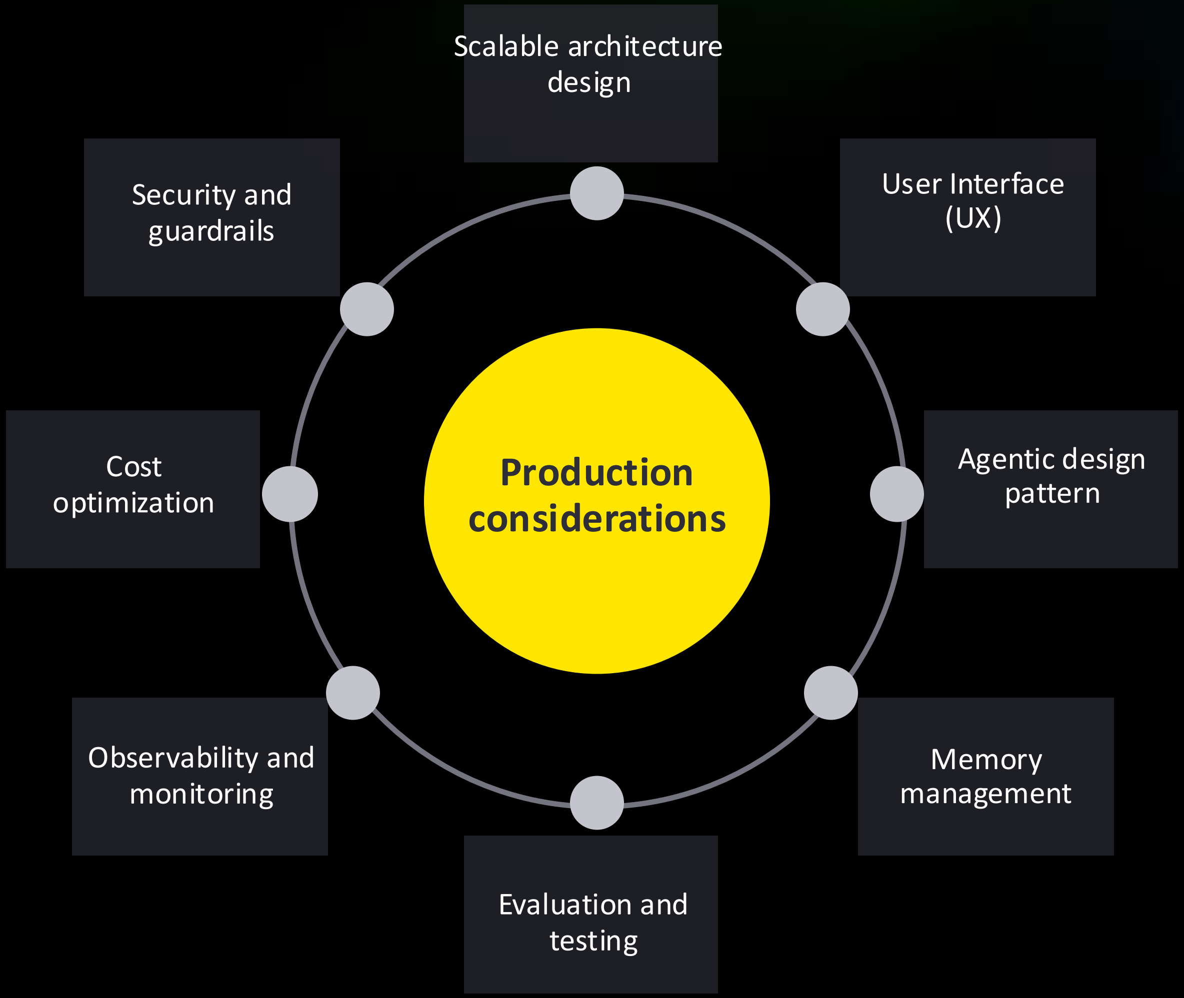
EY

Overview of production challenges in multi-agent system

Multi agentic workflow challenges

Category	Challenge	Mitigation
Design challenges	Limit in no of agent and tools	Clear objectives on task and agent definitions. Prefer code logic over agent
	Evaluation challenges	Define robust evaluation pipeline for faster and holistic debugging and proper feedback loop
	Selection of agent pattern	Avoid complex agent network. Break the Agent flow into specialized workflow and assign skill specific tools
Tool handling challenges	Failure in tools' output	Enforce structured output format. Handle default flow in case of failure
	Vague tools' description and parameters	Ensure well-defined parameters and usage guidelines
	Restrict sensitive or unauthorized access of tools	Define logic for human intervention or blocking user in case of accessing tools
Agent collaboration challenges	Dependency b/w agent's output	Well defined guidelines with dynamic few shots and structured output schema
	Infinite looping	Prefer prompt logic to justify agent's action or reasoning. Define clear termination conditions
	High cost of running	Implement model tiering based on tasks complexity
	Right agent or tool selection	Specialized prompt and hierarchical design
Deployment challenges	Guardrails and security	Rule-based and LLM-based validation. Define human in the loop in case of critical tasks. Prefer default behavior over ambiguity
	Scalability	Scalable architecture for state management, agent calling and low latency
	Handling of failure	Clearly defining default behavior based on recommendation for domain expert

Key production considerations for multi-agent systems



Security and guardrails	▪ Prevent unwanted tool/agent calling and prompt injection ▪ Enforce strict IAM policies to restrict sensitive tools
Cost optimization	▪ Implement token efficiency strategies – caching repeated queries ▪ Model tiering routes tasks for cost appropriate backends
Observability and monitoring	▪ Designing agent and tool specific metrics to assess performance ▪ Tracking and logging of user feedback to further enhance workflow
Evaluation and testing	▪ Robust evaluation pipeline to test E2E workflow and agent trajectory ▪ Metrics to assess tool interactions and failure
Memory management	▪ Leverage different agentic memories to maintain state in agentic system
Agentic design pattern	▪ Selection of suitable agentic pattern
Scalable architecture design	▪ Design infrastructure that scales dynamically and handles fluctuating demands. Horizontal scaling via serverless Kubernetes and vertical scaling by adding multi-region LLM deployment
User interaction	▪ UX design to allow user to interact with agents in human-in-the-loop scenario

Solution architecture design consideration

We recommend step by step analysis of business requirements and then mapping it to various design considerations and non-functional requirements to arrive at optimal multi–Agent solution specifications required

Requirements

Requirement analysis

Step 1: Check if its single step process or sequence of steps

Understand if its functional role-based process or non role based

Is it sequence of AI tasks or autonomy required ?

Design considerations

Level of autonomy and planning

Check autonomy/planning requirements from business process? Is it deterministic.?

Role or task based

- Is it business process with role play like account opening or wealth advisor or is it just series of tasks execution with autonomy

Network pattern

- Determine if we need hierarchical or flat
- Hierarchical layers, collaboration between agents

Performance and info-Sec requirements

- Has client got any agentic framework approved or allows AI autonomous actions
- Latency and performance requirement of use case

Tools and external system integrations

- List tools/function calling - external internet search and other external API's and integrations

Architectural decisions

Functional

LLM selection

Based upon level of reasoning infosec & complexity of tasks

Framework selection

Best fit multi-agent framework based upon design considerations

Layers & hierarchical architecture

Based upon business process complexity and agent interactions

Tools & API's

Use-case based tools, External systems & API

Nonfunctional

Vertical & horizontal scaling

# of Agents, # of tools per Agent, TPM, scalability

Guardrails

Based upon Controls and Risk, select guardrails

Cloud /On-prem

Based upon client hosting & infosec requirement

Monitoring

Define Metrics and Tool

Evaluation

Define evaluation tool & metrics. Automated or manual

Final multi agent configuration

EY

Scalability best practices

Scalability: The ability of AI agents to handle increased workloads, user interactions, and data volumes efficiently without compromising performance or reliability.

Platform level

- Shared memory design to avoid errors (short and long term) enabling agents to coordinate actions & enhance scalability
- Selecting agent network patterns - an agent with too many tools may struggle with decision-making and context management. Use a multi-agent system like LangGraph to solve this
- Scalable agentic platform middleware design - task queues for high request volumes and long-running, stateful agent communication for continuous tasks
- Latency challenges for overall response – Designing agent flows, optimizing prompt caching, load balancing, async tool callings etc to optimize latency
- Horizontal scaling: set up Horizontal Pod Autoscaler (HPA) in case of Kubernetes

Agent level

- Supervisor AI agent capacity to handle volume
- # of Subordinate AI agents in account, in one flow – most frame works have some soft limits on # of agents in flow. For ex. AWS bedrock supports 10 agents per flow and 100 per account, hence right solution design is important
- # of instructions in agent instructions
- Distributed agent caching: cache frequently used data locally for agents to reduce redundant queries & improve speed
- Agent loops - Agents start interacting in loop leading to higher connections and throughput challenges. Use hierarchical agentic architecture, limits, and extensive testing

Tool level

- Retries
- This mechanism allows the system to retry function calls that might have initially failed, improving the robustness of the workflow
- Interrupt feature
- Interrupt function, allows you to pause the execution to gather input from a user and then continue execution
- Asynchronous tool calling
- This helps to trigger multiple functions/Tool calls in parallel for vertical scaling of transactions
- Queue management – Queuing of tasks being routed via tools

Model level

- Increase model invocation capacity with provisioned throughput in Amazon Bedrock, OpenAI
- Use API gateways for traffic spikes in LLM API calls. Multi-region deployment pattern with load balancing to manage TPMS. Or leveraging PTU'
- If underlying LLM is not good at logical breakdown and reasoning, it may select suboptimal agents: Supervisor might assign tasks to less suitable agents (tools), resulting in hallucinated outputs. Use a 'Inspector' agent is this case or select right LLM with reasoning

Challenges in scaling multi framework production deployments

- **Cloud does not support agents in all the regions**
- **Memory of past interactions, state management**
- **Latency**

Limitations at agent level

- **No of calls each agent can handle**
- **No of agents per account**

Limitations at tool level

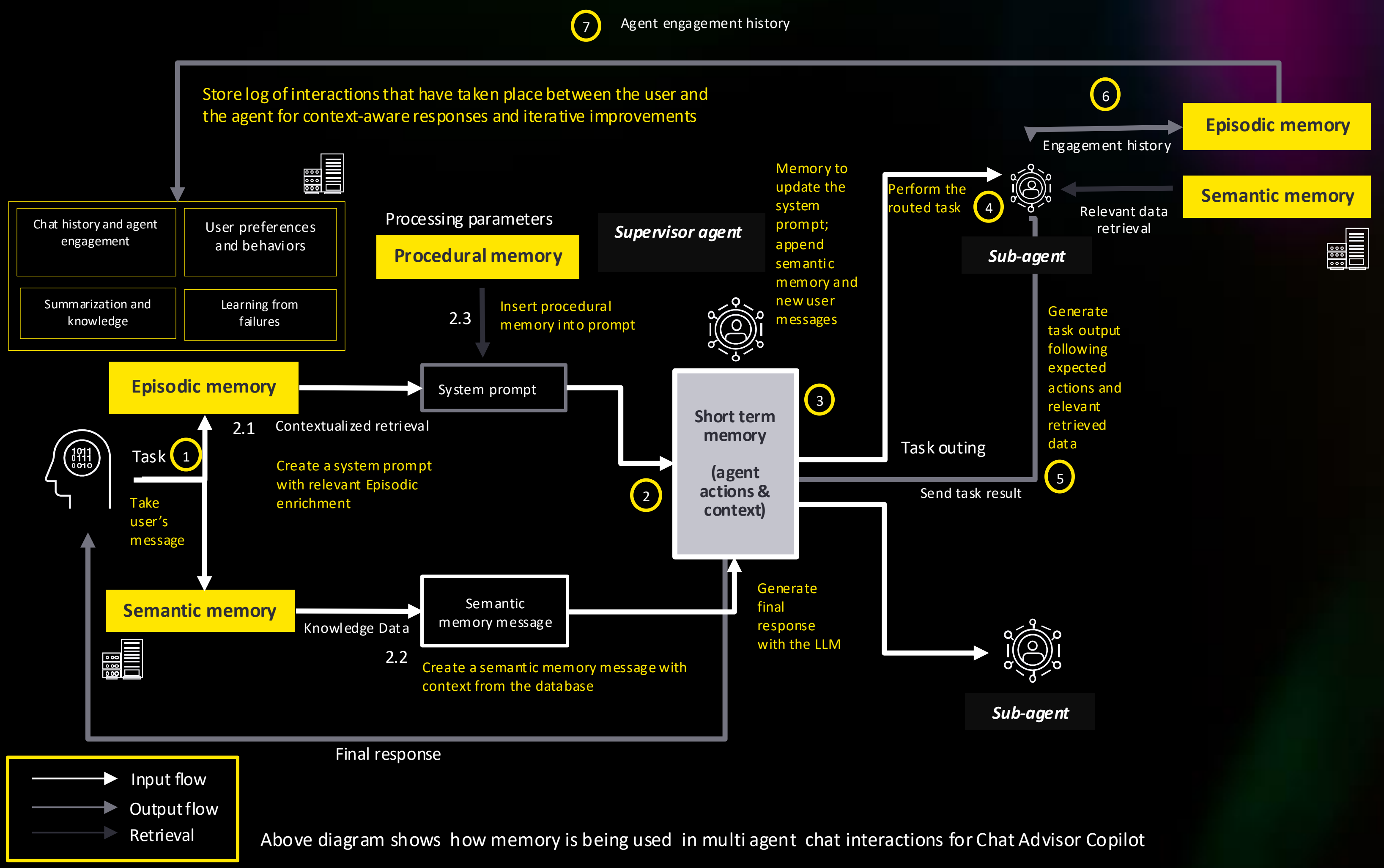
- **Limit on API calls**
- **Tools failures**
- **Tools - Synchronous calls**

Limitations at model/LLM level

- **Context window exceeding**
- **Token limit issue - - No of input and output token processing per minute**

Importance of memory management in agentic workflow

Multi-agent memory management



Above diagram shows how memory is being used in multi agent chat interactions for Chat Advisor Copilot

Short-Term Memory (STM): Temporary storage for immediate tasks like maintaining context of recent conversations optimized for quick access.

In diagram, it is leveraged for storing contextual understanding of conversation across recent chat sessions. (f1 to 30 days)

Long-Term Memory (LTM): Persistent storage for knowledge retention across sessions.

- Episodic: past experiences of users & agents
- Procedural: knowledge of the agent's code,
- Semantic: store general knowledge, facts, and information from various sources

In diagram, it is leveraged to

- Store performance and behaviour of the supervisor agent by maintaining a history of its Reasoning (R), Planning (P), and Tasks (T) etc.
- Stores learnings from failures, user behaviors and preferences, agent history, summarization

Right settings for memory helps in improving accuracy , providing personalized customer experience and improved performance over time.

Key challenges in deploying multi AI-agent solutions

Challenges 1 – Agent trajectory and dependence between agent responses (cont..)

Challenge#1

Understanding agent's trajectory

In multi-agent system, agent usually assigns direct tasks to other different agents. For example, supervisor agent is responsible for assigning tasks to specialized agents.

Our approach

- Add prompt logic to reveal an agent's 'thought process' or reasoning to assign next task using CoT
- Implement dynamic shots to make agent aware of context
- Define explicit rules and logic that determine how the supervisor agent assigns tasks

Challenge#2

High dependencies between each agent's responses

Multi-agent systems depend heavily on agents passing messages and instructions to one another. But what happens when agents misinterpret those instructions, or worse, don't receive them properly at all?

Our approach

- Restrict the response into more reliable structures like json-based requests that help to mitigate miscommunication
- Design single supervisor or workflow engine. This controls message flow or task order
- Designed proper agent memory management so it can help reduce ambiguity when making decisions
- Design business criteria that weigh the importance of each agent's input based on the domain/use-case

Challenge#3

Increased latency due to multiple LLM interactions

When multiple AI agents collaborate (e.g., a supervisor agent coordinating with account opening and wealth advisor agents), sequential LLM queries introduce latency. This can degrade user experience and slow down decision-making.

Our approach

- Use asynchronous programming to handle multiple agent interactions concurrently
- Multi-region LLM deployment and API gateway to ensure availability
- Implement short-term memory caching to store responses from LLMs
- Replace large LLMs with lighter, fine-tuned models for non-complex queries

Multi AI-agent challenges and EY organization’s solution approach

Challenge#4

Handling cases where agent’s response needs improvements

Sometimes an agent may generate incomplete, vague, or incorrect responses. Without quality control, these responses can propagate errors through the system.

Our approach

- Use an independent verification agent to cross-check responses
- Implement self-evaluation prompts where the agent reflects on its own output
- Provide a human override where experts can modify agent outputs before they reach

Challenge#7

Handling unexpected tool responses that could break workflow

In multi-agent system, AI agents rely on external tools (e.g., databases, lookup tools) that may return responses in unexpected formats or fail due to technical issues. If not handled properly, this can break the workflow.

Our approach

- If a tool fails, provide a default response or an alternative action (e.g., retry, switch tools) or delegate the request to an alternative agent or another tool
- If a request fails due to incorrect parameters, an agent can intelligently modify the query and reattempt

Challenge#8

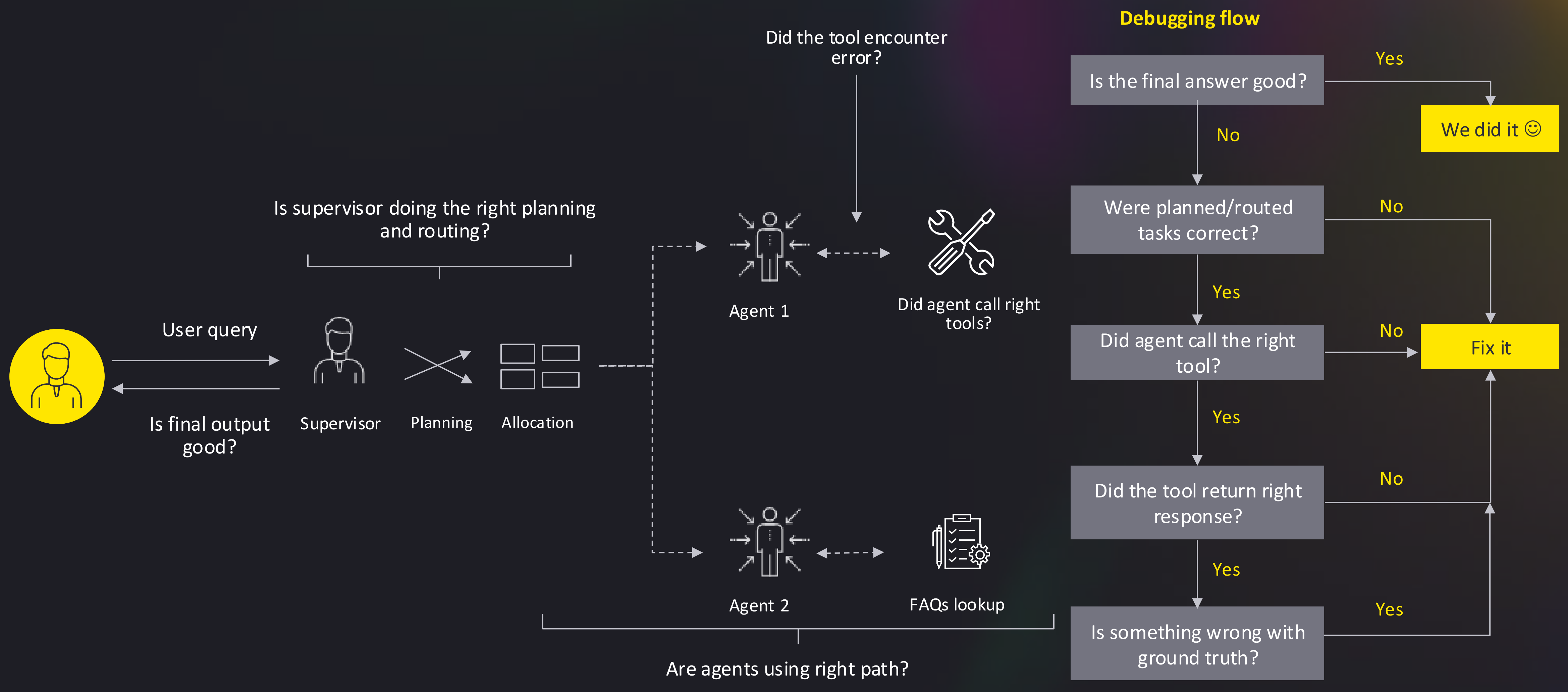
Handling API failures due to asynchronous execution

In multi-agent system, when a supervisor agent assigns a task, subsequent API calls may fail if the agent is still processing the previous request This leads to bottlenecks, dropped tasks, and workflow disruptions.

Our approach

- Instead of waiting for an agent’s response synchronously, supervisor agent should use an async task queue to submit requests and continue processing
- Deploy multiple instances of the same agent and distribute tasks among them to avoid congestion
- If an API call fails due to the agent being busy, implement a circuit breaker mechanism to prevent repeated failures

Multi-agent framework evaluation approach



Evaluating multi-agent framework

It is important to assess overall and individual performance of flow based on application requirements.

AI agent evaluation strategies

1

Define and build golden dataset

To create a comprehensive dataset that enables thorough testing and evaluation of LLM AI agents, ensuring their performance is aligned with real-world applications and challenges.

- Input and expect output (E2E eval)
- Input and expect Trajectory (trajectory eval)
- Input and expect step (step eval)

2

E2E evaluation: to assess the overall performance

Key Metrics:
Correctness, completeness, faithfulness, relevance

Trajectory evaluation: to analyse the sequence of actions by the agent

Key Metrics:
Exact match, in order match, number of extra step, number of mismatched entries, trajectory precision, trajectory recall

Step wise evaluation to break down the agent's performance

Key Metrics:
accuracy, precision, recall

Tool wise evaluation to break down the tools' performance

Key Metrics:
Tool selection, tool error

Evaluation framework (Langgraph, arizeai, aws bedrock agentic eval)	Input (user queries)	Application output/trajectory	Reference output

3

Experiment results

Evaluate the performance and efficiency of the agentic workflow in a high-demand technical environment.

Comparative analysis Setup

Previous run: Compare with previous run to understand the impact.
Incremental feature experiment: Gradual integration of features to isolate impact.
Golden dataset: Compare performance with golden dataset

Agentic evaluation: defining and building evaluation ‘golden’ dataset

An evaluation dataset is a collection of data specifically curated to assess the model's performance. This golden dataset should clearly define the branch user queries, the relevance of the policy/circular to the query and the expected response.

Considerations for building the golden dataset for Evaluation

Initiate incrementally

- Build a manageable scope or subset of data to start
- Include false positive and example of jail-break, prompt injection to evaluate solution’s response
- Start with a smaller dataset to pilot the process and refine methodologies before scaling up

LLM-assist data generation

- Utilize LLM capabilities to analyze policy document to extract key information
- This LLM-assist approach could speed up compilation of evaluation set
- Perform human-based feedback mechanism to validate the quality of generated data with respect to use-case requirements

Expand dataset progressively

- Continuously evaluate and refine the dataset based on feedback and insights gained during initial phases and discussion with branches
- Gradually incorporate additional data sources and variables to enrich the dataset's depth and quality
- **Track user feedback** to append edge cases

Types of queries

Simple query	Reasoning based query	Multi-hop based query
Straightforward query which is direct and does not involve complex reasoning or inference.	Complex inquiries that require logical deduction, inference, or contextual understanding to generate the desired response.	Involves harnessing multiple pieces of information or knowledge sources to answer a complex question

EY

Agentic evaluation: challenges and their mitigation strategy

Challenges with auto–evaluation

Utilizing LLM–as–judge is powerful and can be helpful to automate the evaluation but we have faced challenges around:

1. Most of the evaluation frameworks such as ragas, deepeval, provide metrics with numeric score (out of 10). We observed, the range value has variation in evaluation output

			answer_correctness					
1806_Correct/Wrong /Incomplete		max	min	q25	median	mean	q75	max
Correct	5	1.00	0.21	0.39	0.57	0.55	0.68	0.98
Incomplete	8	1.00	0.18	0.41	0.52	0.51	0.61	0.85
Wrong	6	1.00	0.19	0.25	0.43	0.42	0.50	0.80

2. These frameworks provides some out–of–the–box metrics (faithfulness, relevancy etc.), but was difficult to infer for use–cases
3. Due to non–deterministic nature, multiple runs of LLM–as–judge yields different score/results. Because of which evaluation results are not consistent
4. These evaluation framework has generic prompts and criteria to perform evaluation

Our unique approach/frameworks to solve key challenges

Came up with strategies to mitigate issues and improve the performance of auto–evaluation

1. Modified criteria for evaluation: instead of LLM performing numeric evaluation, we modified prompts to provide binary score based on comparison between Ground Truth data and LLM Output. As mentioned in couple of papers, we also devised criteria to compare two consecutive LLM responses to score
2. Modified evaluation metrics: instead of using typical ragas' metrics, we defined set of metrics i.e., completeness, contradiction etc. to judge LLM response on various aspects
3. Self–consistency (CoT@k) approach to mitigate the variation in LLM evaluation. We performed iteration of evaluation and opted for most frequent response
4. Design domain and use–case specific prompt with few–short steps to allow LLM to improve the evaluation response quality and provide better reasoning to assign specific labels

Guardrails in multi-agentic system

1 Input guardrails

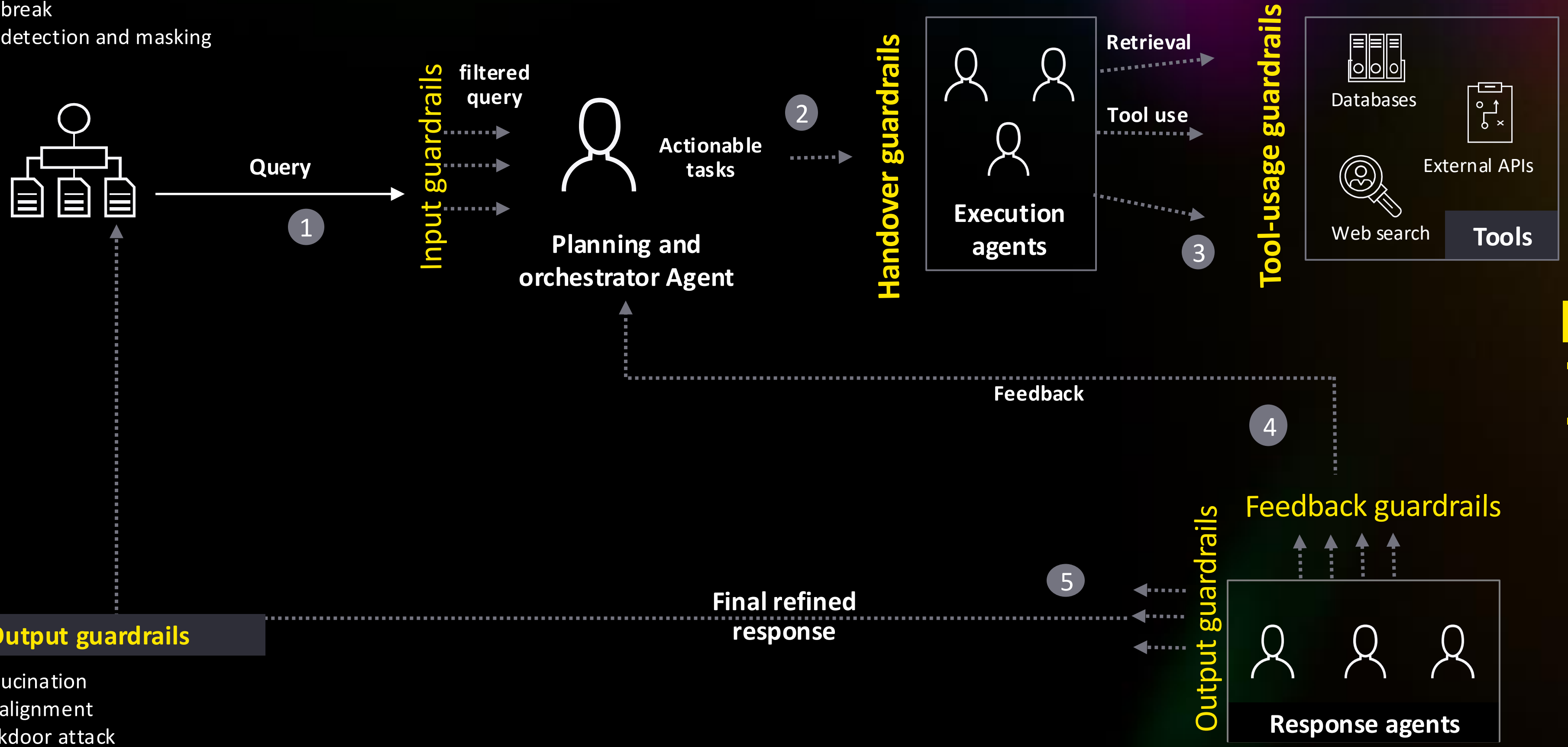
- User authentication
- Pattern detection
- Prompt injection
- Jailbreak
- PII detection and masking

2 Handover guardrails

- Indirect prompt injection
- Planning threat
- Backdoor attacks

3 Tool-use guardrails

- Indirect prompt injection
- Agent2tool threat
- PII detection and masking



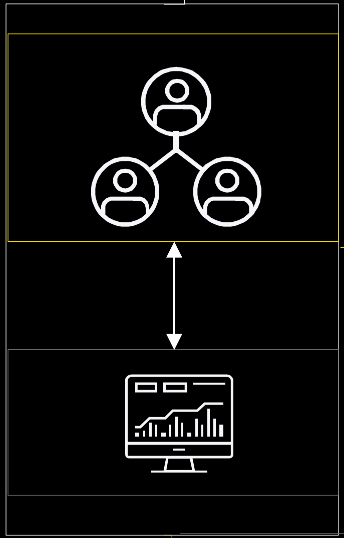
4 Feedback guardrails

- Competition interaction threats
- Cooperative interaction threats

5 Output guardrails

- Hallucination
- Misalignment
- Backdoor attack

Key metrics for agentic AI monitoring



System level

Agent level

Tools level

Guardrails level

Focus on overall application performance and reliability

Cost per task completion
Track cost associated with completion of each task

Latency and throughput
Measure the response time

Token usage per interaction
Measure the number of tokens used in each interaction

Focus on performance and efficiency of agents

Steps per task
Count the number of steps or actions taken by the agent

Task completion Rate
Measure the percentage of tasks completed by the agent

Agent success rate
Track the number of successful task completions related to attempts

API call frequency
Monitor the frequency of API calls made by the agent

LLM call error rate
Measure the error rate of calls made to the language model,

Focus on reliability and robustness of tools

Tool failure rate
Track the rate at which tools are used by the failed agent

Tool execution time
Measure the time spent by the agent's tools in executing tasks

Duplicate tool invocations
Monitor the rate of redundant or duplicate tool usage

Focus on detecting sensitive and harmful content

Prompt injection detection
Identify harmful or sensitive content

Infinite loop detection
Detect if the agent gets stuck in infinite loop

Hallucination detection
Identify hallucination in response

EY | Building a better working world

EY is building a better working world by creating new value for clients, people, society and the planet, while building trust in capital markets.

Enabled by data, AI and advanced technology, EY teams help clients shape the future with confidence and develop answers for the most pressing issues of today and tomorrow.

EY teams work across a full spectrum of services in assurance, consulting, tax, strategy and transactions. Fueled by sector insights, a globally connected, multi-disciplinary network and diverse ecosystem partners, EY teams can provide services in more than 150 countries and territories.

All in to shape the future with confidence.

ED None

GDS Creative Support (GDS CS): CRS_GDS BMC_153240819

ey.com