



CONTEXT CONSTRUCTION VIA **RAG AND AGENT**

Team 6

Fengyu Gao, Shunqiang Feng, Wei Shen, Zihan Zhao



3 PAPERS

- ReAct Framework:

ReAct: Synergizing Reasoning and Acting in Language Models

- Google Vertex AI White Paper on Agents

<https://www.kaggle.com/whitepaper-agents>

- Survey on RAG

Agentic Retrieval-Augmented Generation: A Survey on Agentic RAG



ReAct: Synergizing Reasoning and Acting in Language Models

Zihan Zhao (rxy6cc)

ReAct = Reason + Act

A movie theater screening starts at 2:35 pm. The movie lasts for 2 hours, 32 minutes after 23 minutes of previews. It takes 20 minutes to get home from the movie theater. What time should you tell your family that you'll be home?



Thought for 62 seconds ^

Okay, let's see. The problem is about figuring out what time I'll get home after a movie. The movie starts at 2:35 pm. There are previews first, then the movie itself. Then it takes 20 minutes to get home. I need to add up all these times to find the total time from the start until I arrive home. Let me break it down step by step.

First, the movie screening starts at 2:35 pm. But before the actual movie begins, there are 23 minutes of previews. So the movie itself doesn't start right at 2:35 pm; it starts after the previews. So I need to calculate when the movie starts by adding the preview time to the start time.

So, starting at 2:35 pm, if we add 23 minutes, what time does that get us to? Let me do the math. 2:35 plus 20 minutes would be 2:55 pm, right? Because 35 minutes plus 20 is 55. Then there are 3 more minutes to add to reach 23 minutes total. So 2:55 pm plus 3 minutes is 2:58 pm. So the actual movie starts at 2:58 pm. Got that.

Now, the movie itself lasts 2 hours and 32 minutes. So I need to add 2 hours and 32 minutes to the start time of the movie, which is 2:58 pm. Let's break that down. Adding 2 hours to 2:58 pm would bring us to 4:58 pm. Then we need to add the remaining 32 minutes.

Adding 32 minutes to 4:58 pm. Let's see, 4:58 plus 30 minutes is 5:28 pm, and then adding the remaining 2 minutes would make it 5:30 pm. Wait, no. Wait, 4:58 pm plus 30 minutes is indeed 5:28 pm, and then adding more minutes would be 5:30 pm. Wait, but 32 minutes added to 4:58 pm. Hmm. Let me check again.

DeepSeek is **reasoning** about a grade school math problem



A robot is doing a series of **actions**: squat, roll, jump, etc.

BUT, actions must be in the **text form** for now

An Example

Q: Aside from the Apple Remote, what other device can control the program Apple Remote was originally designed to interact with?

(1d) ReAct (Reason + Act)

Thought 1: I need to search Apple Remote and find the program it was originally designed to interact with.

Act 1: Search[Apple Remote]

Obs 1: The Apple Remote is a remote control introduced in October 2005 by Apple ... originally designed to control the Front Row media center program ...

Thought 2: Apple Remote was originally designed to control the Front Row media center program. I need to search Front Row next and find what other device can control it.

Act 2: Search[Front Row]

Obs 2: Could not find [Front Row]. Similar: ['Front Row Seat to Earth', 'Front Row Motorsports', 'Front Row (software)', ...]

Thought 3: Front Row is not found. I need to search Front Row (software) .

Act 3: Search[Front Row (software)]

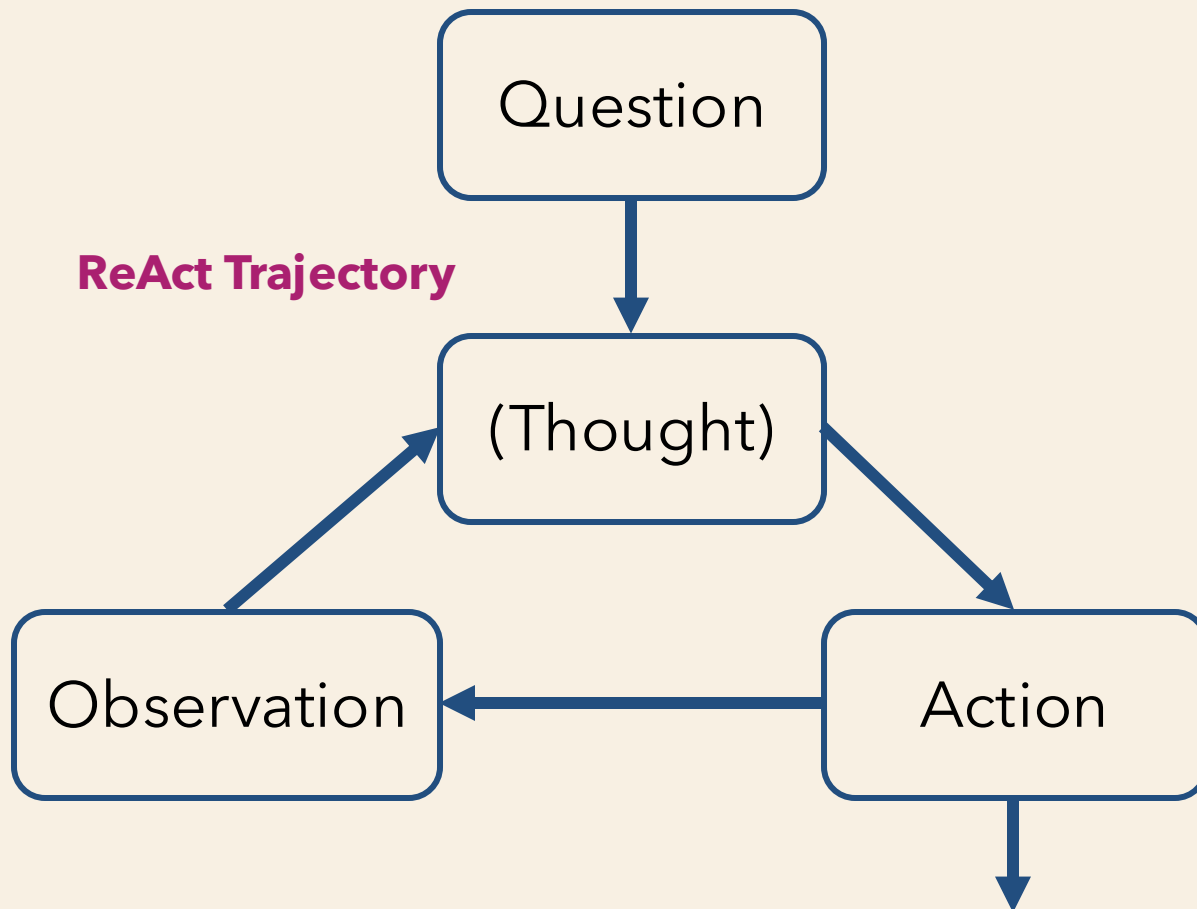
Obs 3: Front Row is a discontinued media center software ...

Thought 4: Front Row (software) is controlled by an Apple Remote or the keyboard function keys. So the answer is keyboard function keys.

Act 4: Finish[keyboard function keys]



Thought-Action-Observation Loop



$$\pi(a_t | c_t),$$
$$c_t = (o_1, a_1, \dots, o_{t-1}, a_{t-1}, o_t)$$

- Time t
- Action a
- Context c
- Observation o



$$\pi(\hat{a}_t | c_t)$$

- Augmented action \hat{a} includes "thoughts"

Results

Prompt Method ^a	HotpotQA (EM)	Fever (Acc)
Standard	28.7	57.1
CoT (Wei et al., 2022)	29.4	56.3
CoT-SC (Wang et al., 2022a)	33.4	60.4
Act	25.7	58.9
ReAct	27.4	60.9
CoT-SC → ReAct	34.2	64.6
ReAct → CoT-SC	35.1	62.0
Supervised SoTA^b	67.5	89.5

Table 1: PaLM-540B prompting results on HotpotQA and Fever.

- **ReAct** → **CoT-SC** when ReAct fails to return an answer within given steps
- **CoT-SC** → **ReAct** when the majority answer among n CoT-SC samples occurs less than $n/2$ times
 - Not confident to do the task with internal knowledge

Results (Limitations)

	Type	Definition	ReAct	CoT
Success	True positive	Correct reasoning trace and facts	94%	86%
	False positive	Hallucinated reasoning trace or facts	6%	14%
Failure	Reasoning error	Wrong reasoning trace (including failing to recover from repetitive steps)	47%	16%
	Search result error	Search return empty or does not contain useful information	23%	-
	Hallucination	Hallucinated reasoning trace or facts	0%	56%
	Label ambiguity	Right prediction but did not match the label precisely	29%	28%

Table 2: Types of success and failure modes of ReAct and CoT on HotpotQA, as well as their percentages in randomly selected examples studied by human.

How is ReAct “learned”?

Original

Question What is the elevation range for the area that the eastern sector of the Colorado orogeny extends into?
Answer 1,800 to 7,000 ft



ReAct

Question What is the elevation range for the area that the eastern sector of the Colorado orogeny extends into?
Thought 1 I need to search Colorado orogeny, find the area that the eastern sector of the Colorado orogeny extends into, then find the elevation range of the area.
Action 1 Search[Colorado orogeny]
Observation 1 The Colorado orogeny was an episode of mountain building (an orogeny) in Colorado and surrounding areas.
Thought 2 It does not mention the eastern sector. So I need to look up eastern sector.
Action 2 Lookup[eastern sector]
Observation 2 (Result 1 / 1) The eastern sector extends into the High Plains and is called the Central Plains orogeny.
Thought 3 The eastern sector of Colorado orogeny extends into the High Plains. So I need to search High Plains and find its elevation range.
Action 3 Search[High Plains]
Observation 3 High Plains refers to one of two distinct land regions
Thought 4 I need to instead search High Plains (United States).
Action 4 Search[High Plains (United States)]
Observation 4 The High Plains are a subregion of the Great Plains. From east to west, the High Plains rise in elevation from around 1,800 to 7,000 ft (550 to 2,130 m).[3]
Thought 5 High Plains rise in elevation from around 1,800 to 7,000 ft, so the answer is 1,800 to 7,000 ft.
Action 5 Finish[1,800 to 7,000 ft]

- Randomly select a couple of cases from HotpotQA and Fever datasets
 - 6 cases from HotpotQA
 - 3 cases from Fever
- Manually compose ReAct-format trajectories to use as few-shot exemplars

ReAct is ...

- A prompting method
- A few-shot learning technique
- Based on Chain-of-Thought (CoT)

- Intuitive
- Easy to use
- Flexible

ReAct is NOT ...

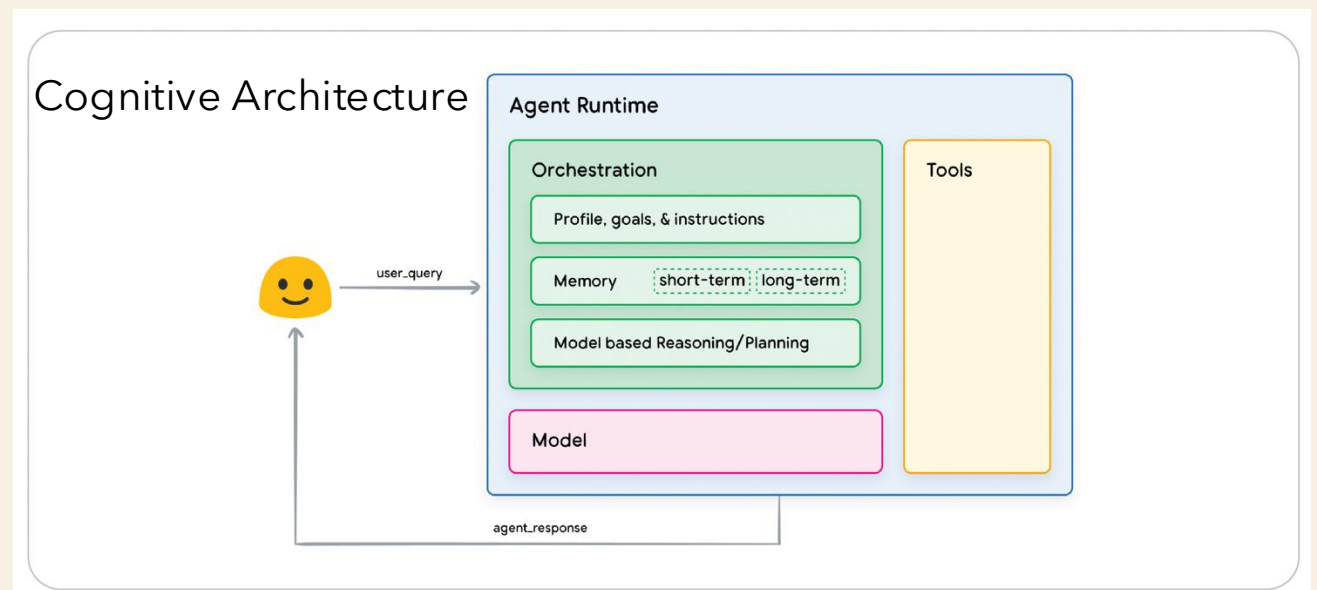
- Just CoT
- A replacement for RL
- Free from error propagation



White Paper - Agents

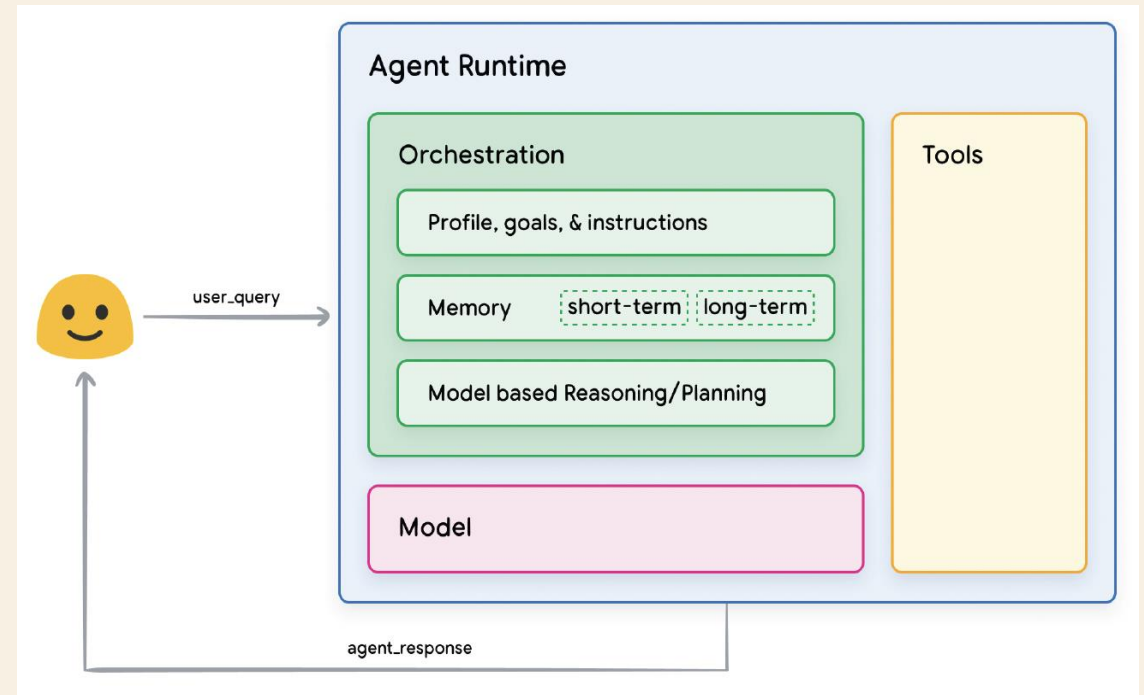
What is an Agent?

- A Generative AI agent is an application that attempts to achieve a goal by observing the world and acting upon it using tools that it has at its disposal.
 - Autonomous
 - Independent
 - Proactive
 - General



Cognitive Architecture

- Model: One or more language models
- Tools: Bridge to connect to the external world
- Orchestration: Action planning

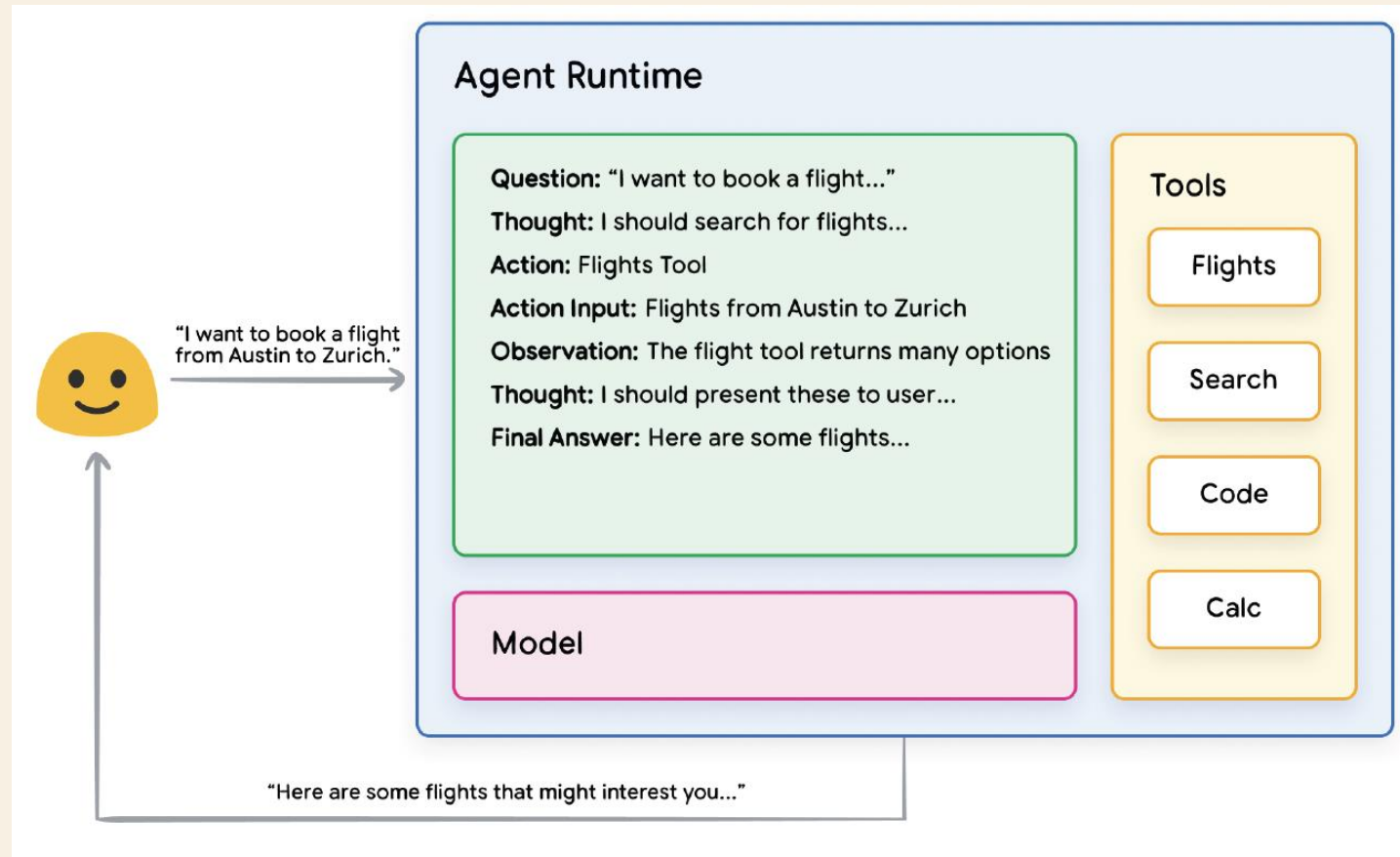


A Comparison

- Agents have
 - Extended knowledge
 - Both long-term and short-term memory
 - Natively supported tools
 - Thinking

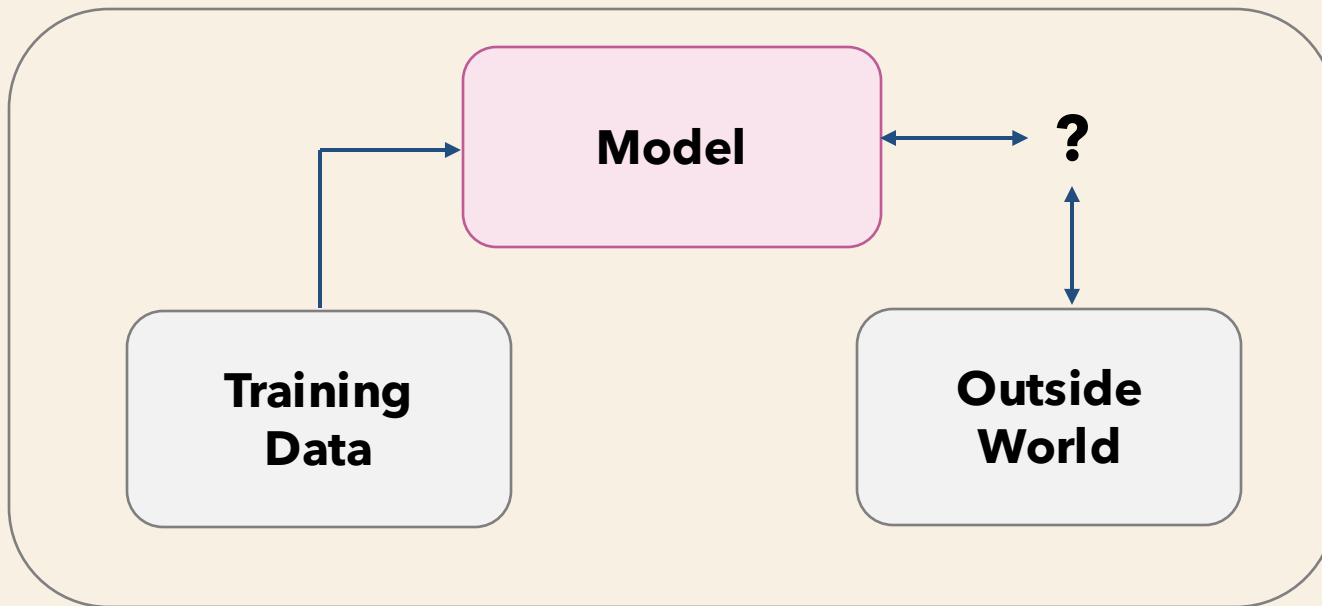
Models	Agents
Knowledge is limited to what is available in their training data.	Knowledge is extended through the connection with external systems via tools
Single inference / prediction based on the user query. Unless explicitly implemented for the model, there is no management of session history or continuous context. (i.e. chat history)	Managed session history (i.e. chat history) to allow for multi turn inference / prediction based on user queries and decisions made in the orchestration layer. In this context, a 'turn' is defined as an interaction between the interacting system and the agent. (i.e. 1 incoming event/ query and 1 agent response)
No native tool implementation.	Tools are natively implemented in agent architecture.
No native logic layer implemented. Users can form prompts as simple questions or use reasoning frameworks (CoT, ReAct, etc.) to form complex prompts to guide the model in prediction.	Native cognitive architecture that uses reasoning frameworks like CoT, ReAct, or other pre-built agent frameworks like LangChain.

Agent Workflow



Shunqiang Feng (mpp7ez)

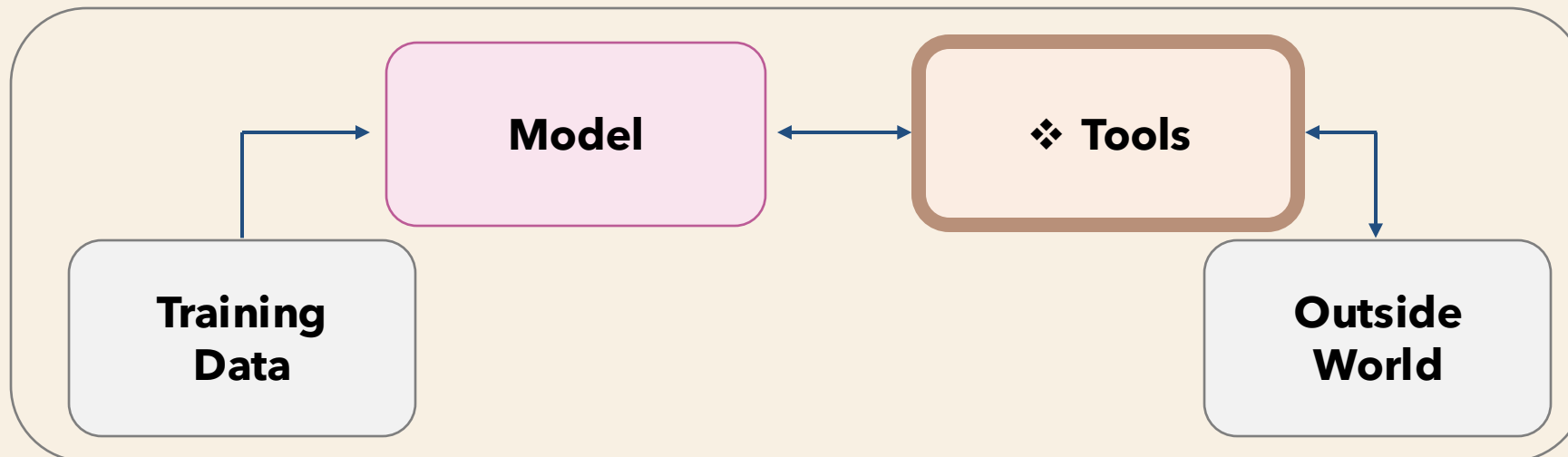
Tools: keys to the outside world



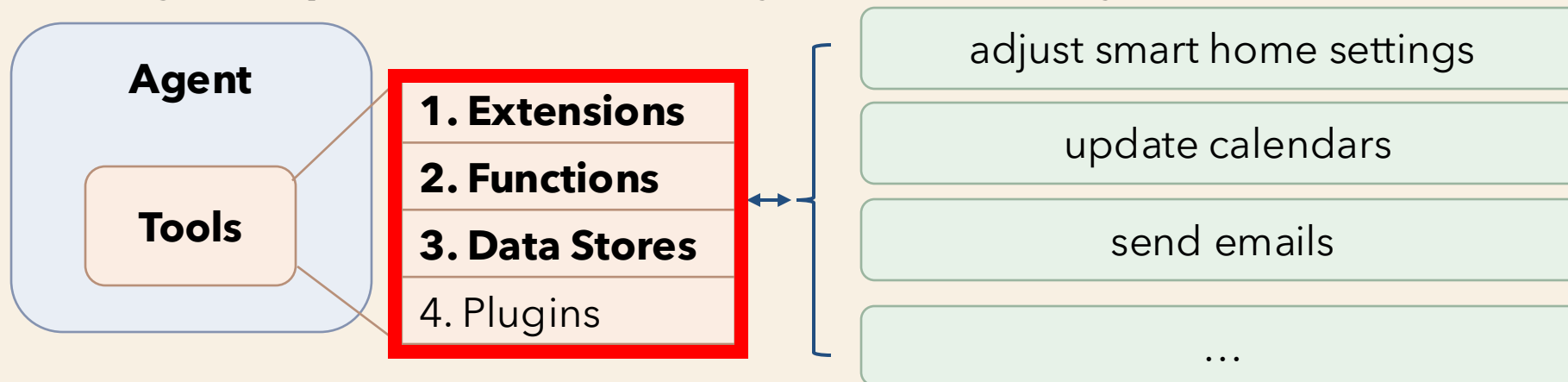
Regardless of how much data we throw at a model, it still lacks the fundamental ability to **interact** with the outside world, which means, the model lacks the ability to directly **perceive and influence** the real world.

Google Vertex AI Agent **Tools:** **keys to the outside world**

- **Tools are what create a link between our foundational models and the outside world**



- **Tools enable the agent to perform a wider variety of tasks in 4 ways**



Google Vertex AI Tool1. Extensions

Bridging Agents & APIs

- **What are Extensions?**
 - Standardized connectors between **Agents & APIs**
 - Enable seamless API execution **without custom code**
- **Why use Extensions?**

Think about this scenario: in a flight booking use case, a user might state "I want to book a flight *from Austin to China.*"

✗ Without Extensions:

- Requires **custom code** for extracting flight details.
- **Fails** if key info (e.g., departure city) is missing.
- Hard to scale across **different API** providers.

✓ With Extensions:

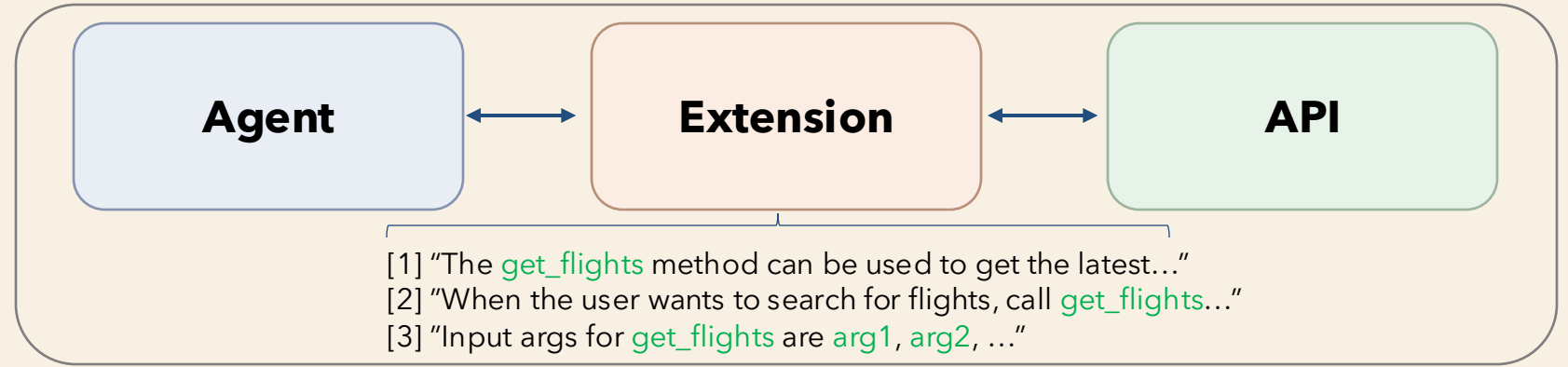
- **Automates API interactions** without extra coding.
- Handles missing information **intelligently**.
- **Scalable** for different airlines & travel services.

Tool1. Extensions

Bridging Agents & APIs

- **How Extensions Work?**

1. Teach the agent how to use APIs with **examples**.
2. Define necessary **parameters** for successful API calls.



- **Dynamic Selection of Extensions**

- ✓ **Independent & Configurable**

Extensions are crafted separately but included in the agent's setup.

- ✓ **Smart Selection at Runtime**

The agent analyzes user queries and picks the best-fit Extension.

- ✓ **Built-in Example Types**

Enables dynamic adaptation, improving flexibility & accuracy.

Tool1. Extensions

Bridging Agents & APIs

- **How Agents Select the Right Extension**

Just like a **developer** picks the right API, an **agent** selects the best **Extension** for a query.

- **Examples:**

✈️ **Flight Booking?** → Uses **Google Flights API**

☕ **Find a Coffee Shop?** → Uses **Google Maps API**

- 💡 **Try it in Gemini**

Go to **Settings > Extensions**, enable Google Flights, and ask:
"Show me flights from Austin to China next Friday."

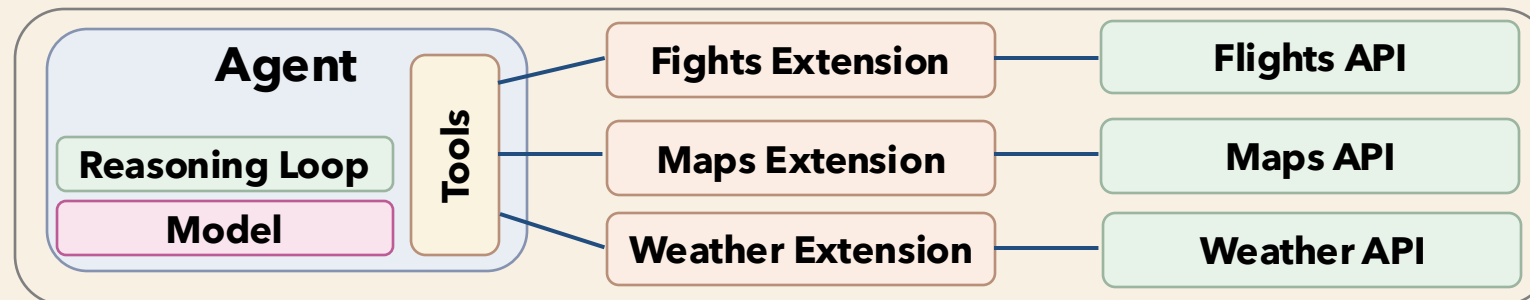


Figure: 1-to-many relationship between Agents, Extensions & APIS

Google Vertex AI Tool2. Functions

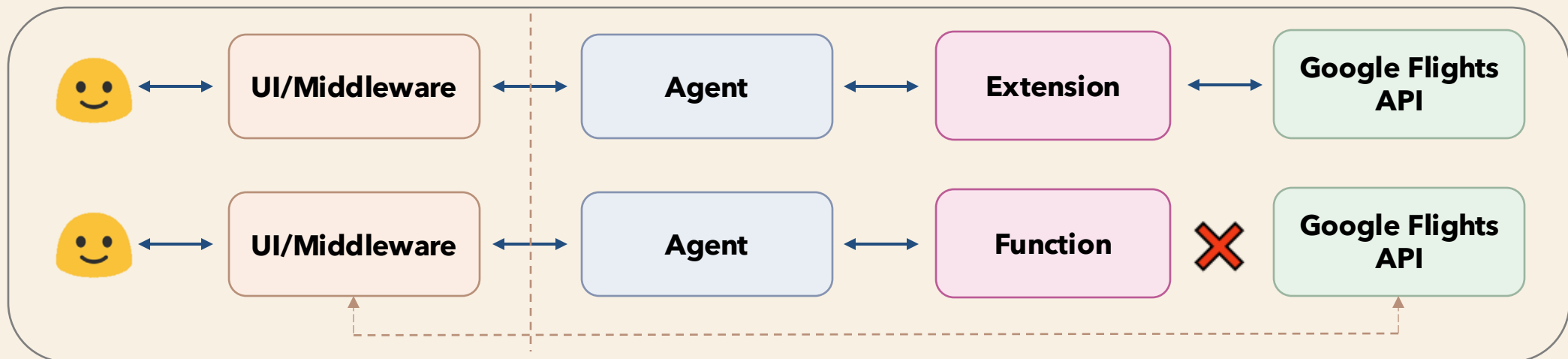
Client-side API execution

- **Starting with Software Engineering**

- Like in Software Engineering, functions are defined as **self-contained modules of code that accomplish a specific task** and can be **reused** as needed.
- In the world of agents, instead of **the software developer**, functions are created by a **language model**.

- **Difference from Extensions**

- Model outputs a function & arguments but **does not make live API calls**.
- **Client-side execution**, not agent-side.



Tool2. Functions

Client-side API execution

- **Functions are needed when:**
 - API calls need to happen **outside the agent's architecture** (e.g., middleware, frontend).
 - **Security restrictions** prevent direct agent API calls.
 - **Timing constraints** require delayed or batch processing ...
- **Key Points of Function Calling**
 - Function calling gives developers **greater control** over API execution and overall data flow.
 - The **developer decides** the best approach based on the specific application needs. For example, the developer may choose **not** to return API response data to the agent since it was irrelevant to future actions.

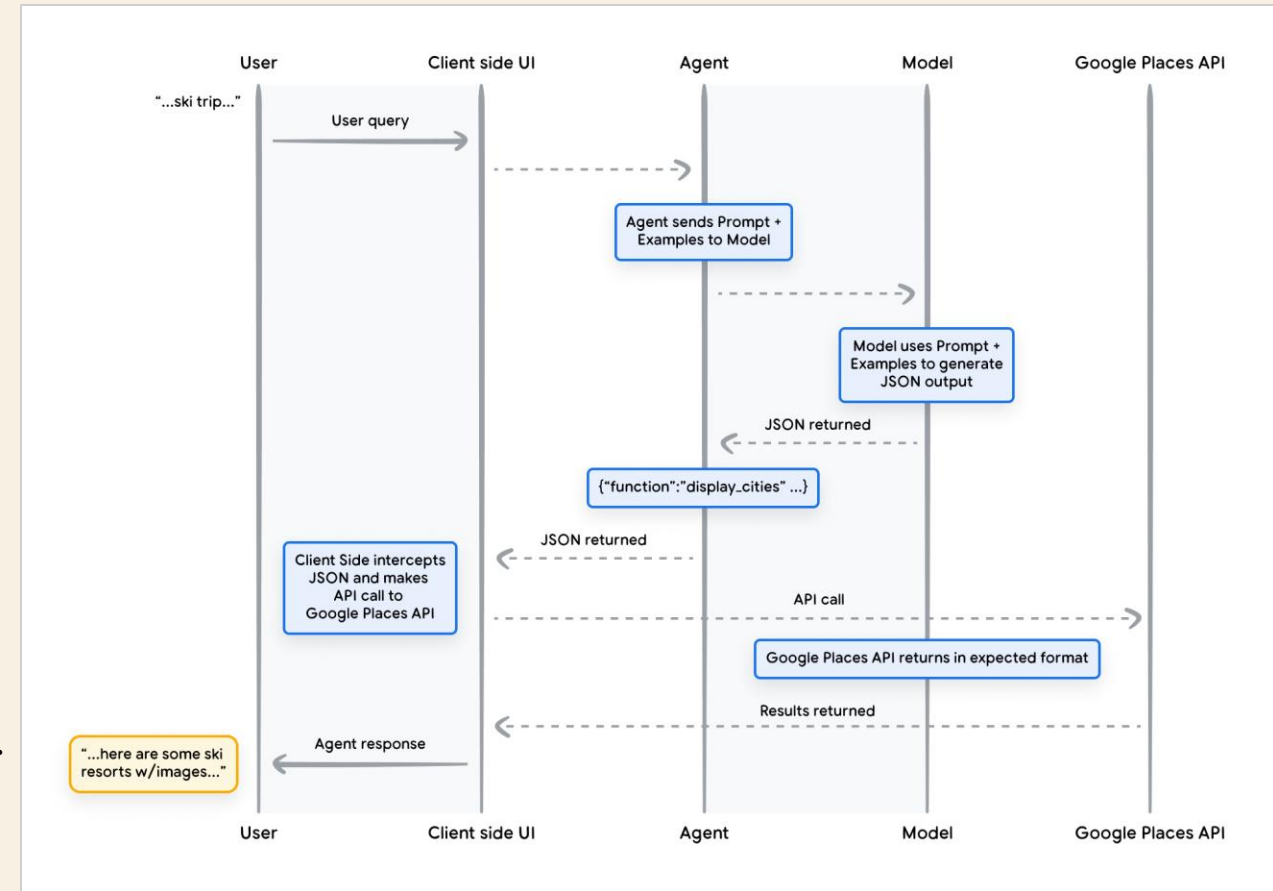
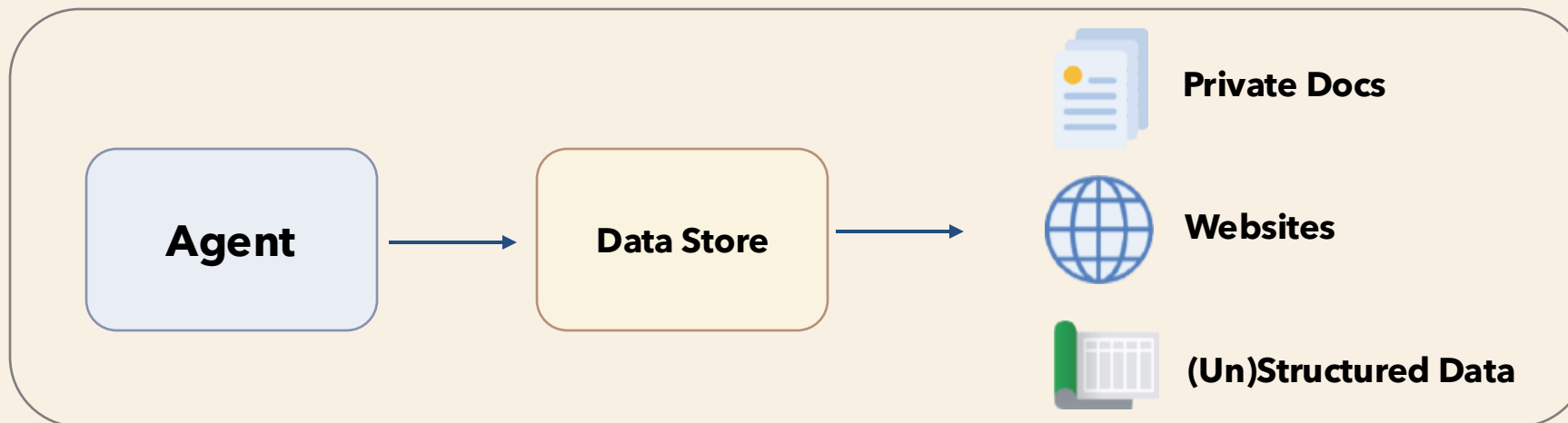


Figure: A sample usage of Function

Google Vertex AI Tool3. Data Store

Dynamic knowledge storage

- **Why we need Data Store in LLM**
 - The model doesn't acquire new information after training
 - However, Real-world knowledge evolves constantly
 - As a result, the model's knowledge becomes outdated
- **Data Store: providing access to dynamic, up-to-date information**
 - Data Stores allow developers to add data in its original format **without retraining the model**.
 - The data is converted into **vector embeddings**, which the agent uses to improve its responses.



Tool3. Data Store

Dynamic knowledge storage

- **Retrieval Augmented Generation (RAG)**
 - One of the most prolific examples of Data Store usage with language models in recent times has been the implementation of RAG based applications.
 - These applications seek to **extend the breadth and depth of a model's knowledge beyond the foundational training data** by giving the model access to data in various formats like:
 - Website content
 - Structured Data in formats like PDF, Word Docs, CSV, Spreadsheets, etc.
 - Unstructured Data in formats like HTML, PDF, TXT, etc.

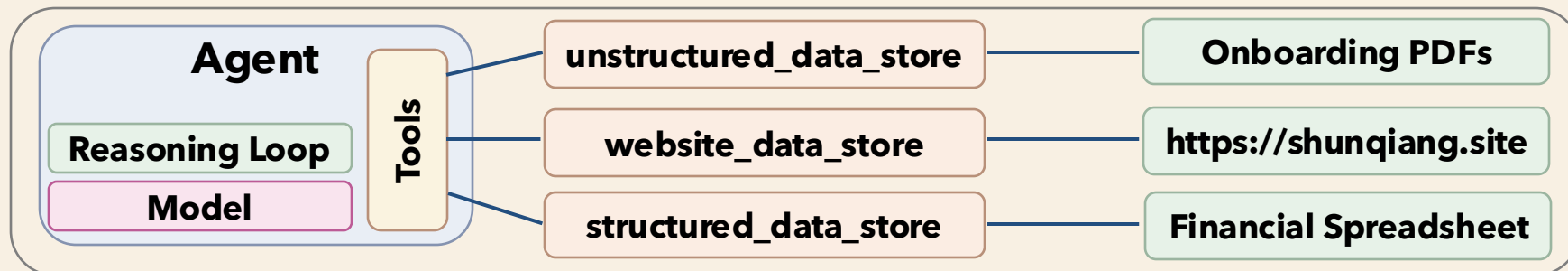
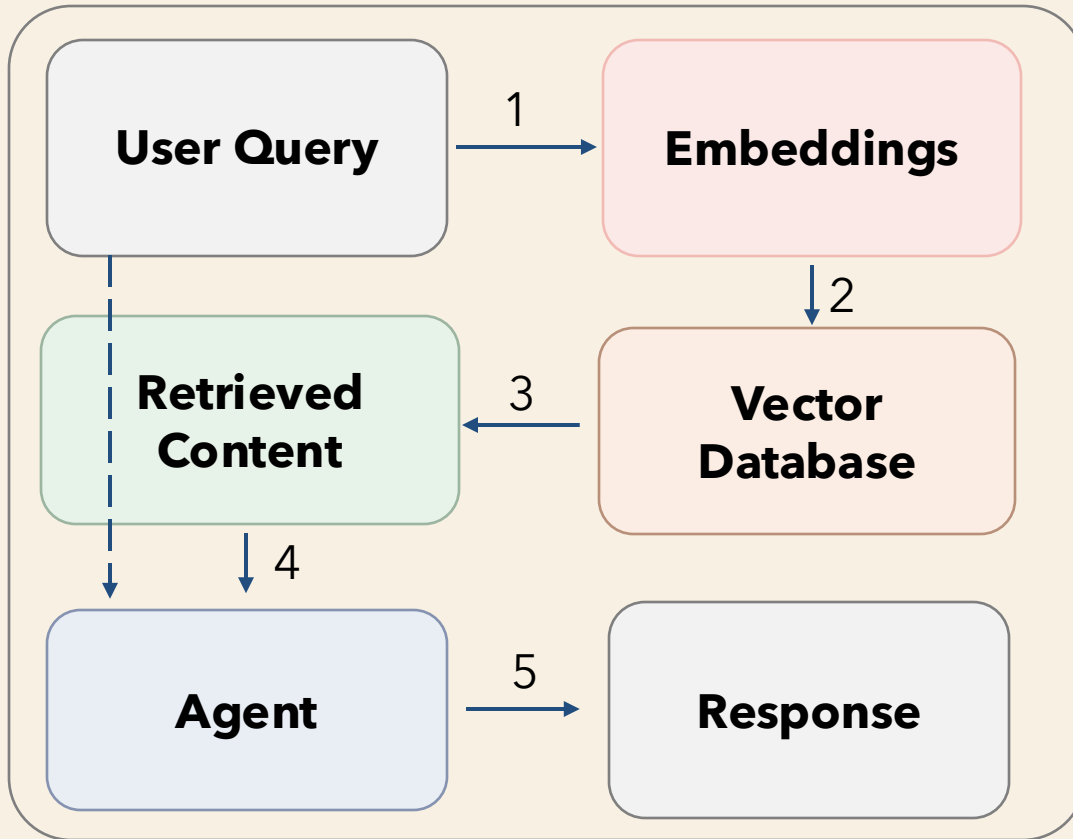


Figure: 1-to-many relationship between Agents and data stores

Tool3. Data Store

Dynamic knowledge storage

- **Retrieval Augmented Generation (RAG)**



1. A user query is sent to an embedding model to generate embeddings for the query
2. The query embeddings are then matched against the contents of the vector database using a matching algorithm like SCaNN
3. The matched content is retrieved from the vector database in text format and sent back to the agent
4. The agent receives both the user query and retrieved content, then formulates a response or action
5. A final response is sent to the user

Figure: The lifecycle of a RAG based application

Tool3. Data Store

Dynamic knowledge storage

- **Retrieval Augmented Generation (RAG)**

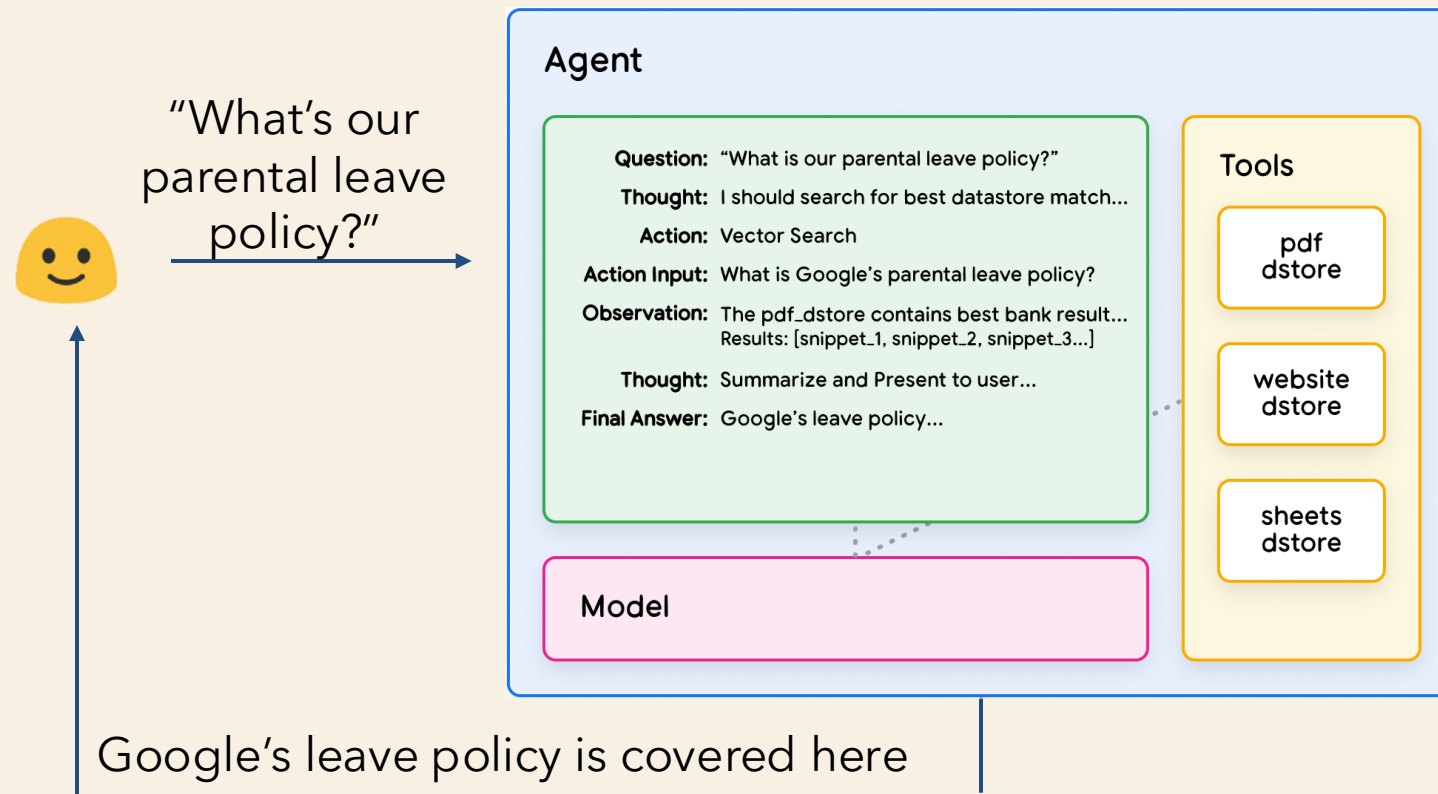


Figure: Sample RAG based application w/ ReAct reasoning/planning

Google Vertex AI Agent **Tools Recap**

	Extensions	Functions	Data Stores
Execution	Agent-Side	Client-Side	Agent-Side
Use Case	<ul style="list-style-type: none">○ Agent manages API calls.○ Uses built-in tools (e.g., Vertex Search).○ Supports multi-step API actions.	<ul style="list-style-type: none">○ API blocked by security/auth.○ Timing/order constraints (e.g., batch jobs).○ API not internet-accessible.	<ul style="list-style-type: none">○ Supports RAG.○ Uses website data.○ Handles structured files (PDF, Word, CSV).○ (Non-) Relational Databases○ Reads unstructured text (HTML, TXT).

Targeted learning

❑ Approaches to enhance model's performance in specific task.

1. In-Context Learning 📌 [Example: ReAct framework]

Uses prompts, tools, and few-shot examples **at inference time** & Learns '**on the fly**' how to use tools for specific tasks.

◆ *Analogy:* A chef receives a recipe, key ingredients, and sample dishes, then figures out the dish in real-time.

2. Retrieval-Based In-Context Learning 🔍 [Example: RAG-based architectures]

Dynamically retrieves relevant information from **external memory** & Enhances model by pulling in **real-time knowledge**.

◆ *Analogy:* A chef selects ingredients and cookbooks from a stocked pantry to refine a dish.

3. Fine-Tuning Based Learning 🎯 [Best for: Domain-specific knowledge enhancement]

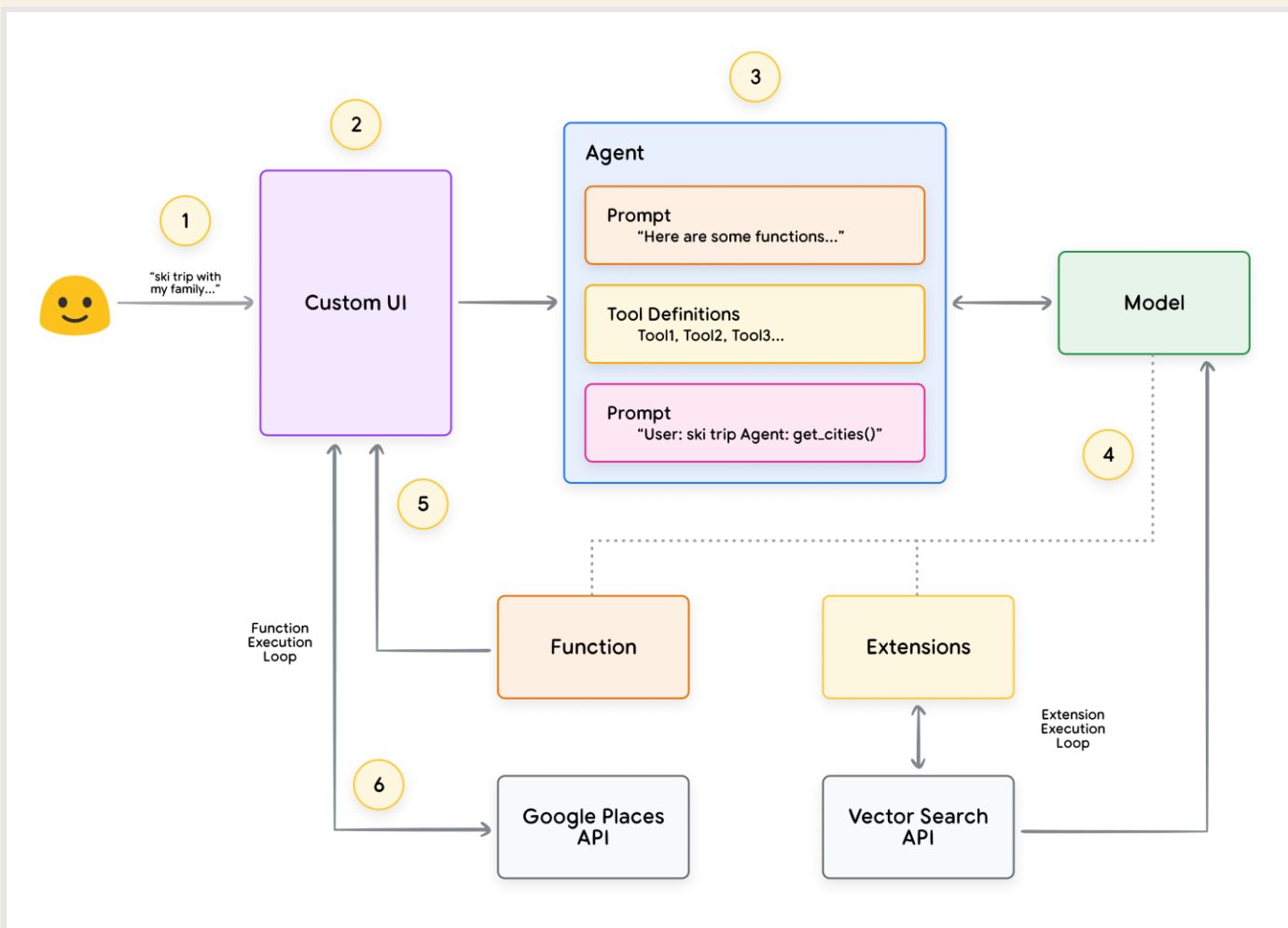
Trains the model with a larger dataset **before inference** & Helps the model understand tool usage **before receiving queries**.

◆ *Analogy:* A chef attends culinary school to master a cuisine before serving customers.

❑ Optimized Performance via Hybrid Approach

By integrating these approaches, we enhance model adaptability, ensuring efficient and scalable AI performance.

Vertex AI Simplifies Agent Creation



Highlights:

- **Simplified Development:** Vertex AI integrates agents with tools for UI, evaluation, and continuous improvement.
- **Natural Language Interface:** Developers can define agent goals, tasks, tools, and sub-agents quickly.
- **Development Tools:** Includes testing, evaluation, debugging, and performance measurement tools.
- **Production-Ready Architecture:** Built using features like Vertex Agent Builder and Vertex Extensions.

Summary of Google Agents Whitepaper

1. Agent Capabilities

- Extend language models with real-time tools and complex task execution.

2. Orchestration Layer

- Structures reasoning, planning, and decision-making using techniques like **ReAct** and **Chain-of-Thought**.

3. External Tools

- **Extensions:** Bridge agents to external APIs, enabling real-time data access.
- **Functions:** Parameter generation for client-side API execution.
- **Data Stores:** Access structured/unstructured data for data-driven applications.

4. Future of Agents

- Increasing reasoning capabilities & **Agent Chaining** empower agents to solve complex problems.
- Building complex agent architectures requires **iterative experimentation and refinement**.

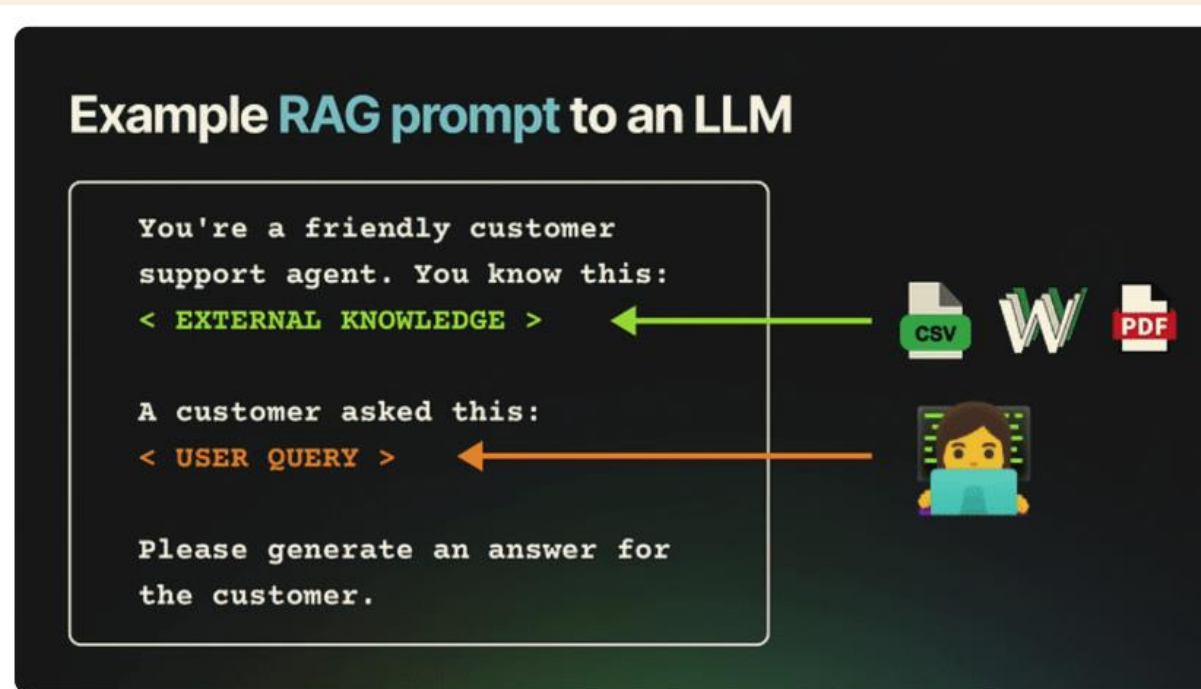


Agentic Retrieval-Augmented Generation: A Survey on Agentic RAG

Fengyu Gao (wan6jj)

Retrieval-Augmented Generation (RAG)

Overview: RAG systems combine the capabilities of LLMs with **retrieval mechanisms** to generate contextually relevant and accurate responses.



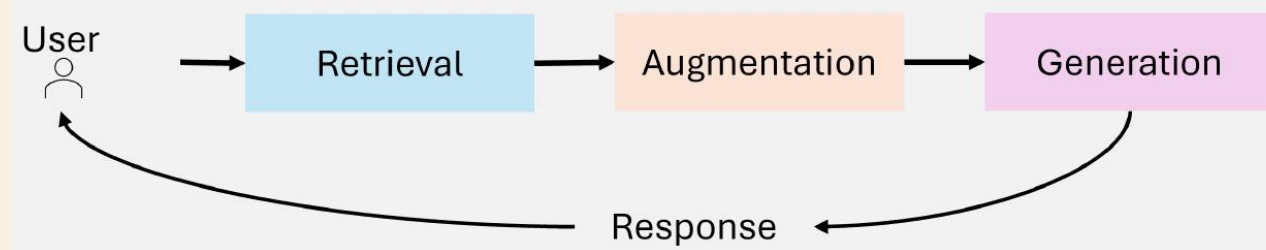
Source: [How to feed LLMs with data from the web](#) (WebExpo 2024)

Retrieval-Augmented Generation (RAG)

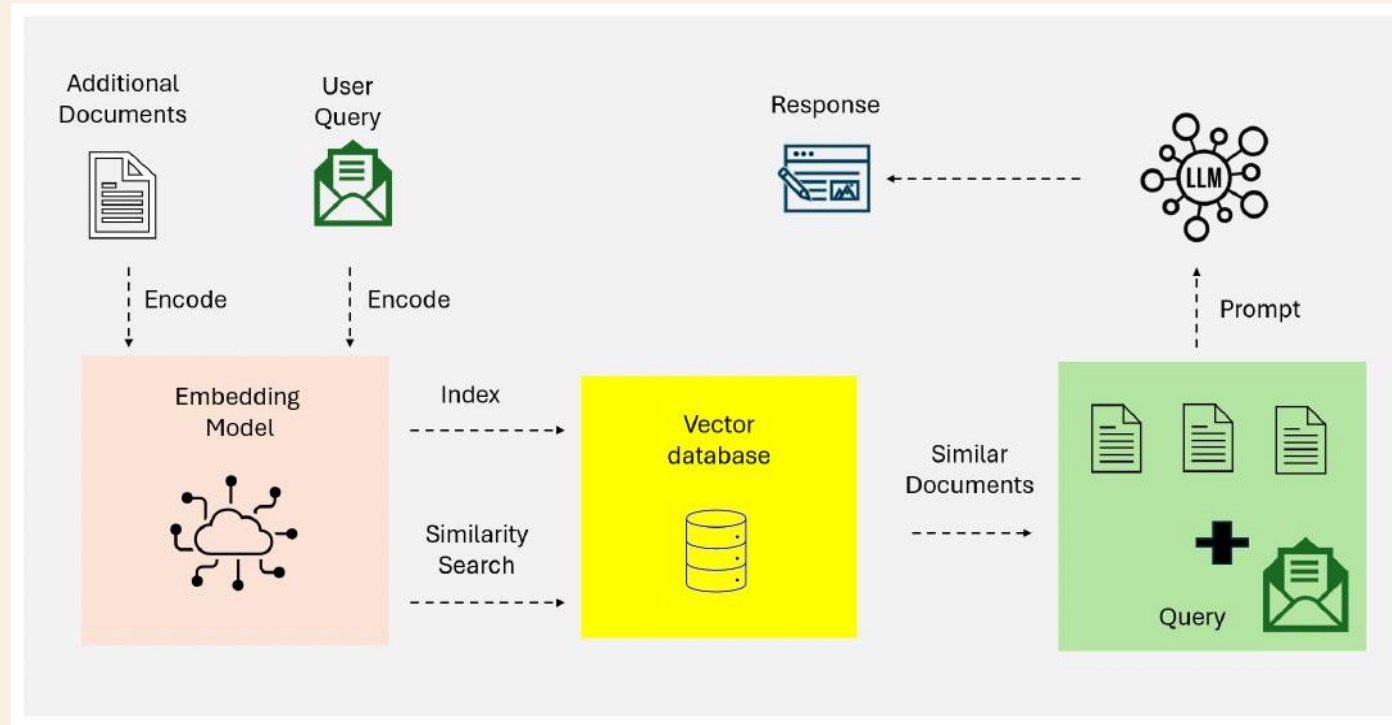
Overview: RAG systems combine the capabilities of LLMs with **retrieval mechanisms** to generate **contextually relevant and accurate responses**.

Core Components:

1. **Retrieval:** Query external sources using advanced search techniques.
2. **Augmentation:** Extract and summarize the most relevant information to align with query context.
3. **Generation:** Combine retrieved data with LLM knowledge for coherent responses.



Example of traditional RAG



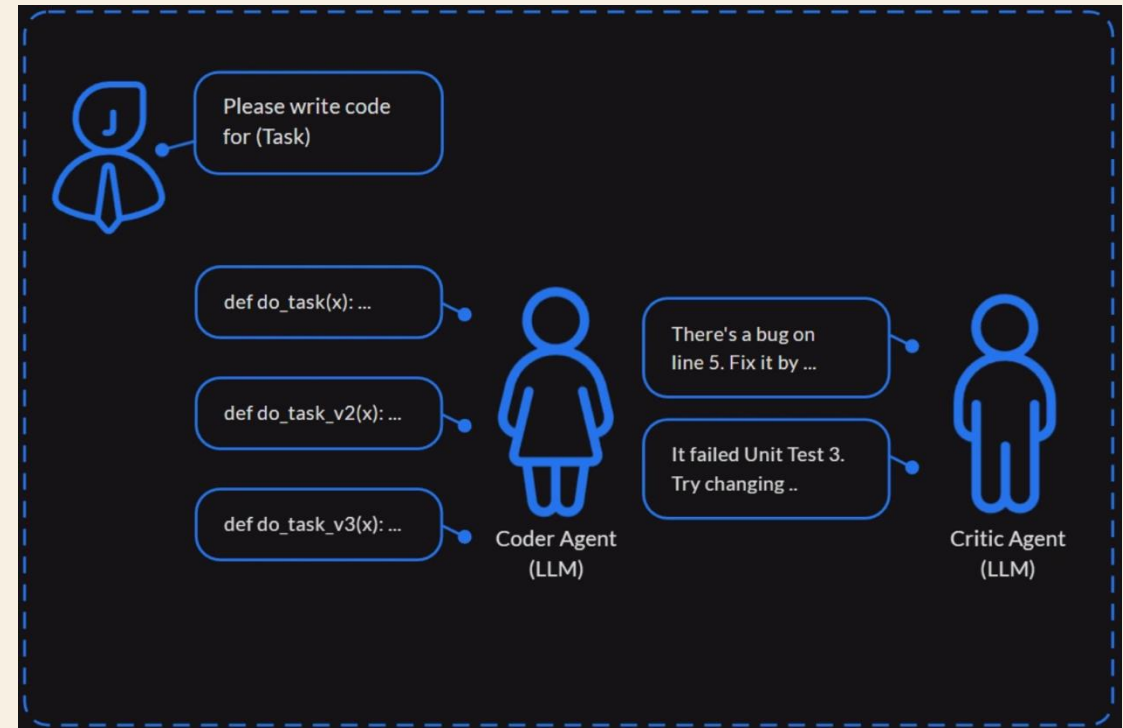
Limitations: contextual integration, multi-step reasoning, and scalability and latency issues.

Agentic RAG

Overview: Agentic Intelligence enables RAG to handle complex reasoning tasks, adapt to dynamic environments, and collaborate effectively.

Agentic Patterns:

1. Reflection
2. Planning
3. Tool Use
4. Multi-Agent Collaboration



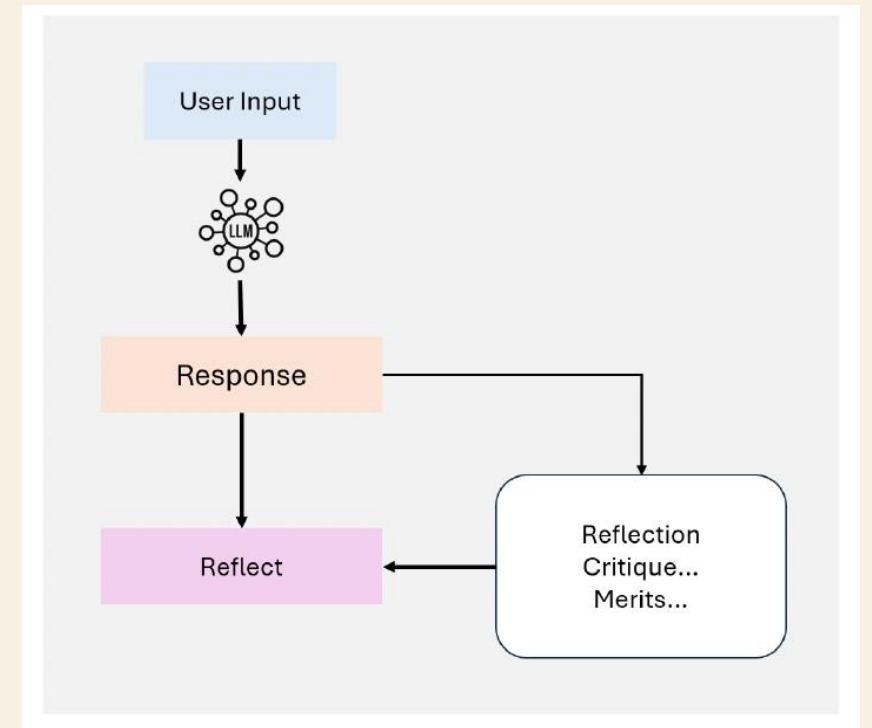
1. Reflection

Definition: Agents evaluate their own decisions and outputs, identifying errors and areas for improvement.

Benefits:

- Enables iterative refinement of results.
- Enhances accuracy in multi-step reasoning tasks.

Example: In a medical diagnostic system, agents refine diagnoses based on iterative feedback from retrieved data.



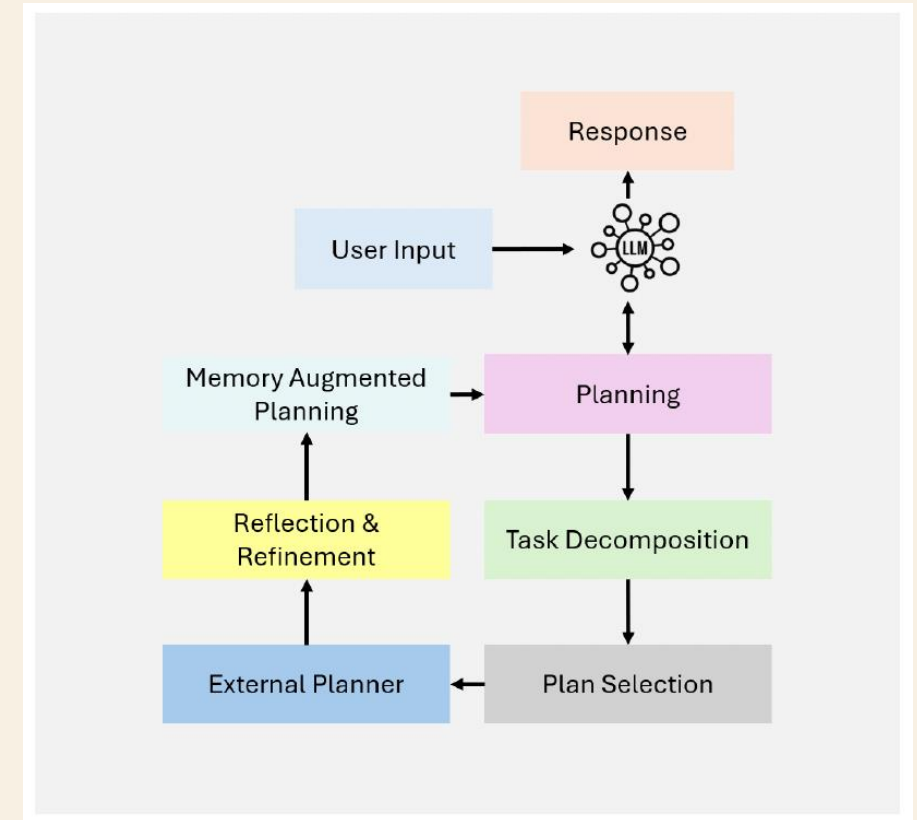
2. Planning

Definition: Agents create structured workflows and task sequences to solve problems efficiently.

Benefits:

- Facilitates multi-step reasoning by breaking down tasks.
- Reduces computational overhead through optimized task prioritization.

Example: A financial analysis system retrieves relevant market data, analyzes risks step by step, and then generates recommendations.



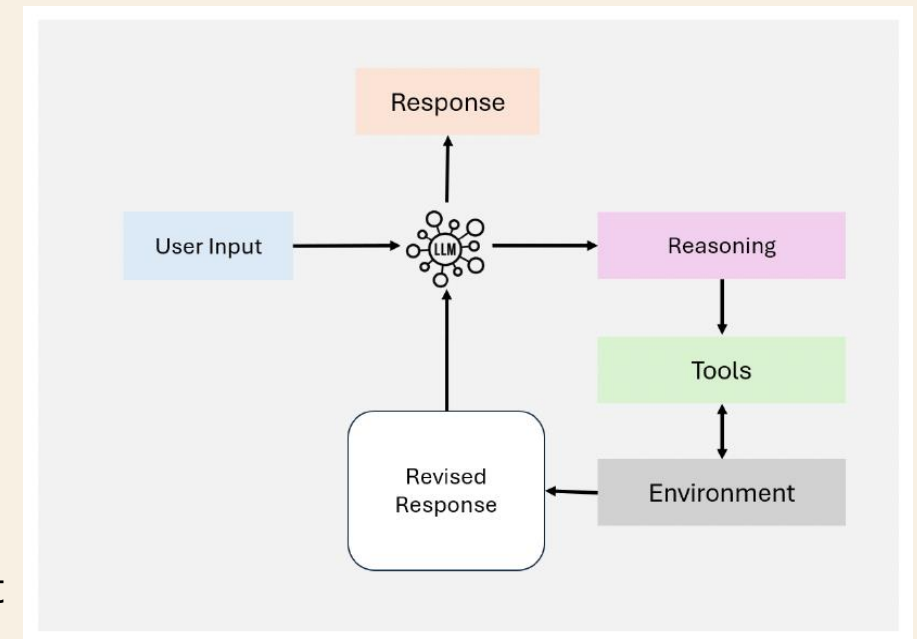
3. Tool Use

Definition: Agents interact with external tools, APIs, and knowledge bases to retrieve and process data.

Benefits:

- Expands the system's capabilities beyond pre-trained knowledge.
- Enables domain-specific applications by integrating external resources.

Example: A legal assistant agent retrieves clauses from contract databases and applies domain-specific rules for compliance analysis.



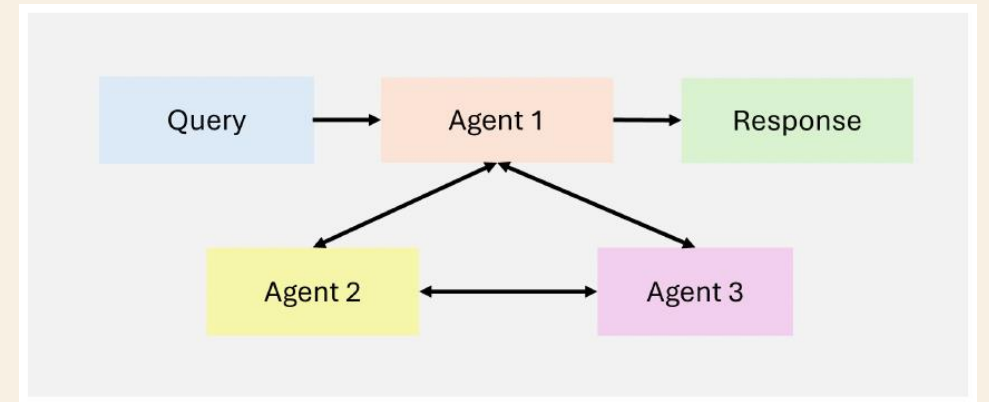
4. Multi-Agent Collaboration

Definition: Multiple agents collaborate to divide and conquer complex tasks, sharing information and results.

Benefits:

- Handles large-scale and distributed problems efficiently.
- Combines specialized agent capabilities for better outcomes.

Example: In customer support, agents collaborate to retrieve knowledge from FAQs, generate responses, and provide follow-ups.



Agentic Workflow Patterns

Overview: Agentic workflow patterns help structure LLM-based applications to optimize performance, accuracy, and efficiency. Different approaches are suitable depending on task complexity and processing requirements.

Approaches:

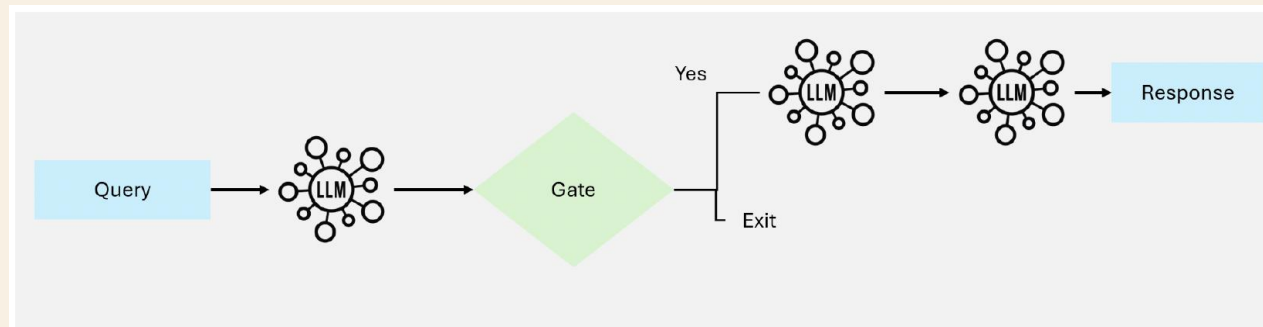
1. Prompt Chaining: Enhancing Accuracy Through Sequential Processing
2. Routing: Directing Inputs to Specialized Processes
3. Parallelization: Speeding Up Processing Through Concurrent Execution
4. Orchestrator-Workers: Dynamic Task Delegation
5. Evaluator-Optimizer: Refining Output Through Iteration

1. Prompt Chaining

Definition: Prompt chaining decomposes a complex task into multiple steps, where each step builds upon the previous one. This approach improves accuracy by simplifying each subtask before moving forward, but may increase latency due to sequential processing.

When to Use: A task can be broken down into fixed subtasks, each contributing to the final output, e.g., step-by-step reasoning.

Example: Structuring document creation by first generating an outline, verifying its completeness, and then developing the full text.

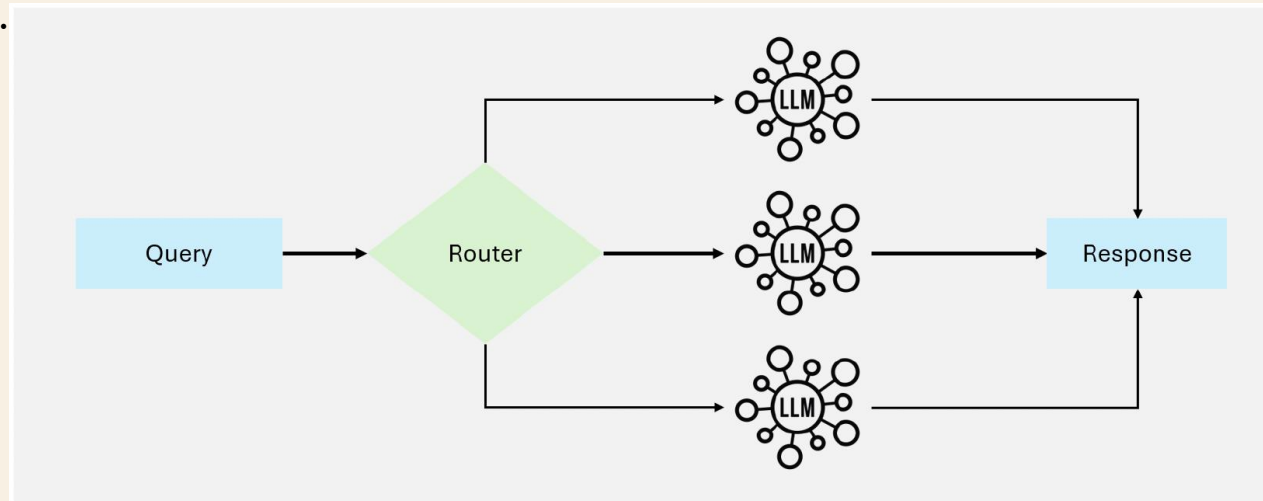


2. Routing

Definition: Routing classifies inputs and directs them to specialized prompts or processes, ensuring distinct tasks are handled separately for better efficiency and response quality.

When to Use: Different types of input require distinct handling strategies, ensuring optimized performance for each category.

Example: Assigning simple queries to smaller models for cost efficiency, while complex requests go to advanced models.



3. Parallelization

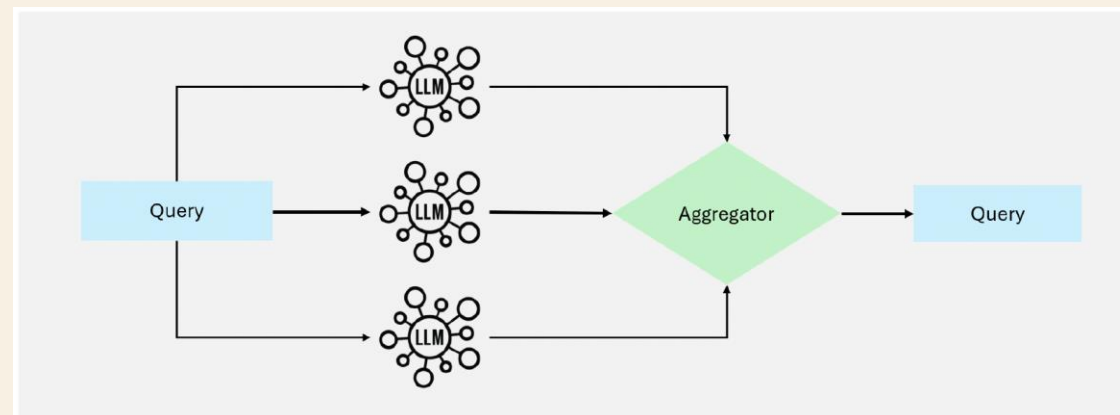
Definition: Parallelization divides a task into independent processes that run simultaneously, reducing latency and improving throughput. It can be categorized into sectioning (independent subtasks) and voting (multiple outputs for accuracy).

When to Use: Useful when tasks can be executed independently to enhance speed or when multiple outputs improve confidence.

Example:

Sectioning: Splitting tasks like content moderation, where one model screens input while another generates a response.

Voting: Using multiple models to cross-check code for vulnerabilities or analyze content moderation decisions.

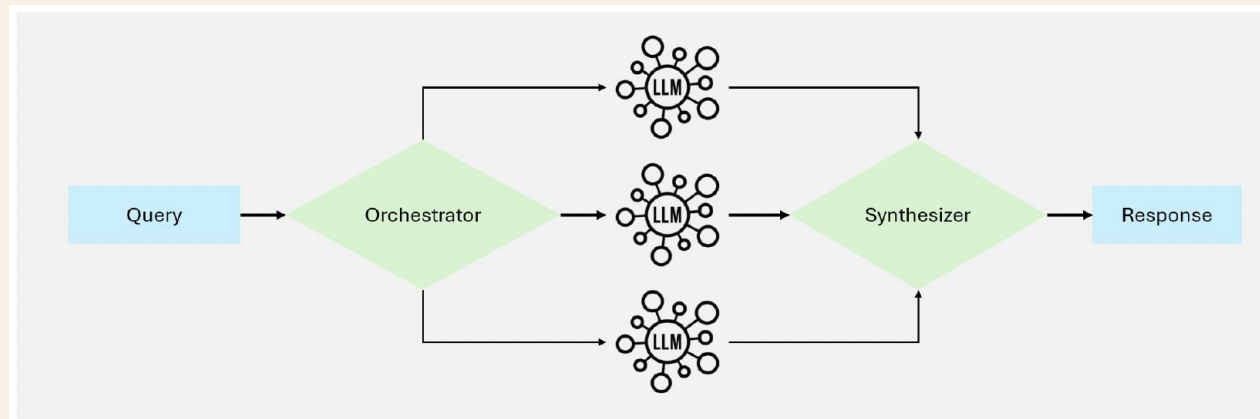


4. Orchestrator-Workers

Definition: This workflow features a central orchestrator model that dynamically breaks tasks into subtasks, assigns them to specialized worker models, and compiles the results. Unlike parallelization, it adapts to varying input complexity.

When to Use: Best suited for tasks requiring dynamic decomposition and real-time adaptation, where subtasks are not predefined.

Example: Automatically modifying multiple files in a codebase based on the nature of requested changes.

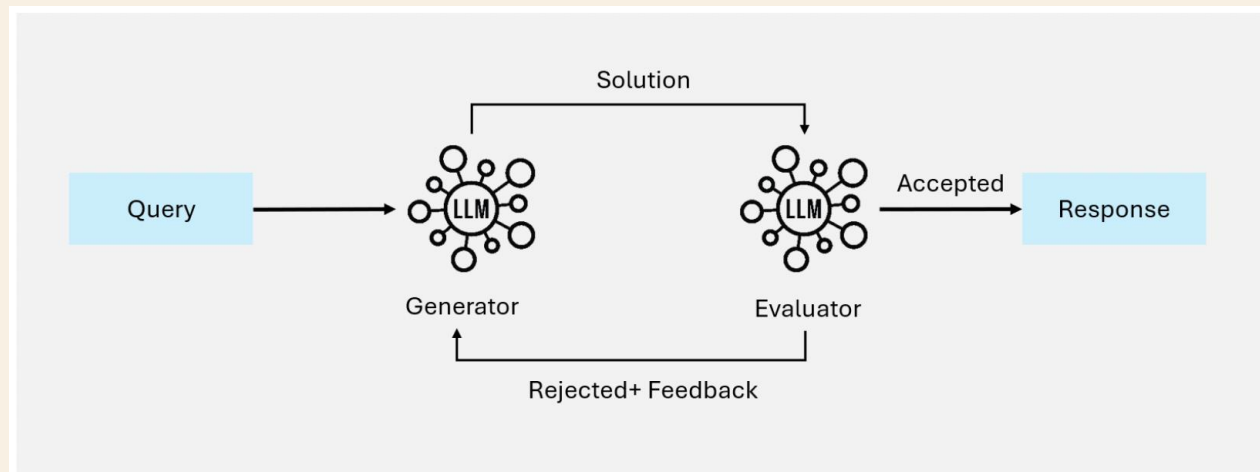


5. Evaluator-Optimizer

Definition: The evaluator-optimizer workflow iteratively improves content by generating an initial output and refining it based on feedback from an evaluation model.

When to Use: Effective when iterative refinement significantly enhances response quality, especially when clear evaluation criteria exist.

Example: Improving literary translations through multiple evaluation and refinement cycles.



Wei Shen (zyy5hb)

Agenda

- **Taxonomy of Agentic RAG Systems**
- **Applications of Agentic RAG**

RAG(retrieval-augmented generation)

Example RAG prompt to an LLM

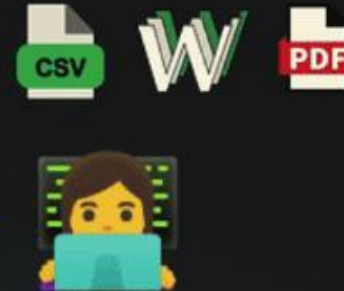
You're a friendly customer support agent. You know this:

< EXTERNAL KNOWLEDGE >

A customer asked this:

< USER QUERY >

Please generate an answer for the customer.



Source: [How to feed LLMs with data from the web](#) (WebExpo 2024)

Taxonomy of Agentic RAG Systems

- 1. Single-Agent RAG**
- 2. Multi-Agent RAG**
- 3. Hierarchical Agentic RAG**
- 4. Corrective Agentic RAG**
- 5. Adaptive Agentic RAG**
- 6. Graph-Based Agentic RAG**

Single-Agent RAG

Key Idea: A **single** autonomous agent manages the retrieval and generation process.

Workflow:

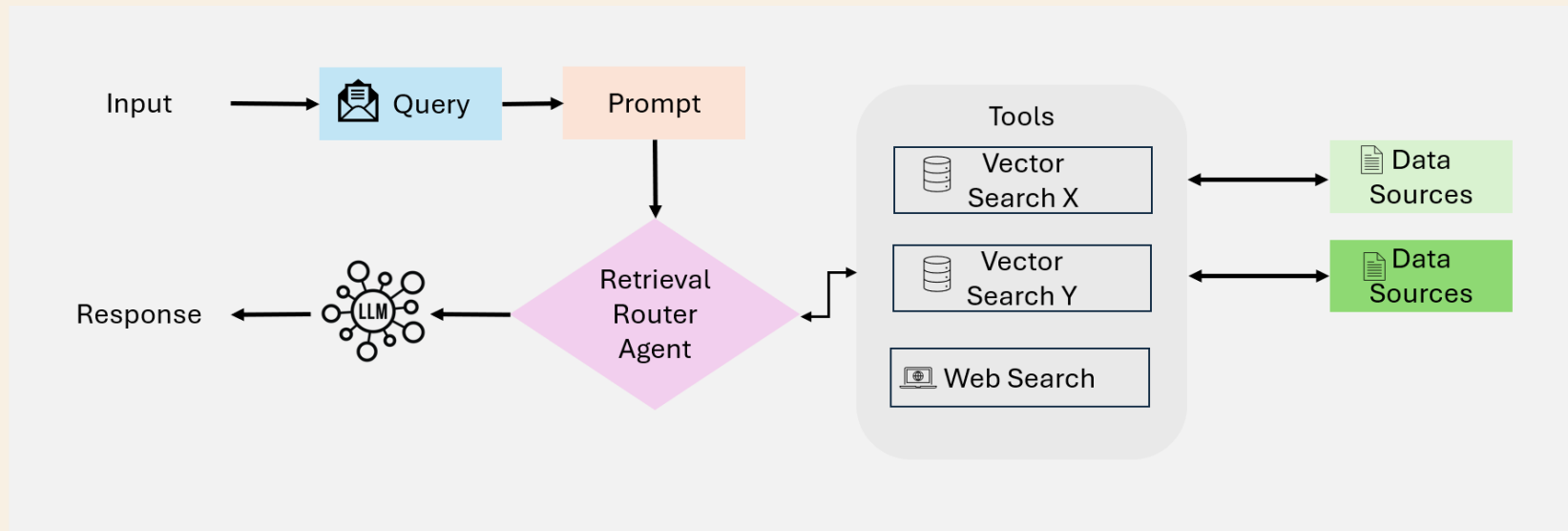
- Query is submitted to the agent.
- The agent retrieves relevant data from external sources.
- Data is processed and synthesized into a response.

Advantages:

- Simple architecture for basic use cases.
- Easy to implement and maintain.

Limitations:

- Limited scalability.
- Ineffective for multi-step reasoning or large datasets.



Multi-Agent RAG

Key Idea: A **team** of agents collaborates to perform complex retrieval and reasoning tasks.

Workflow:

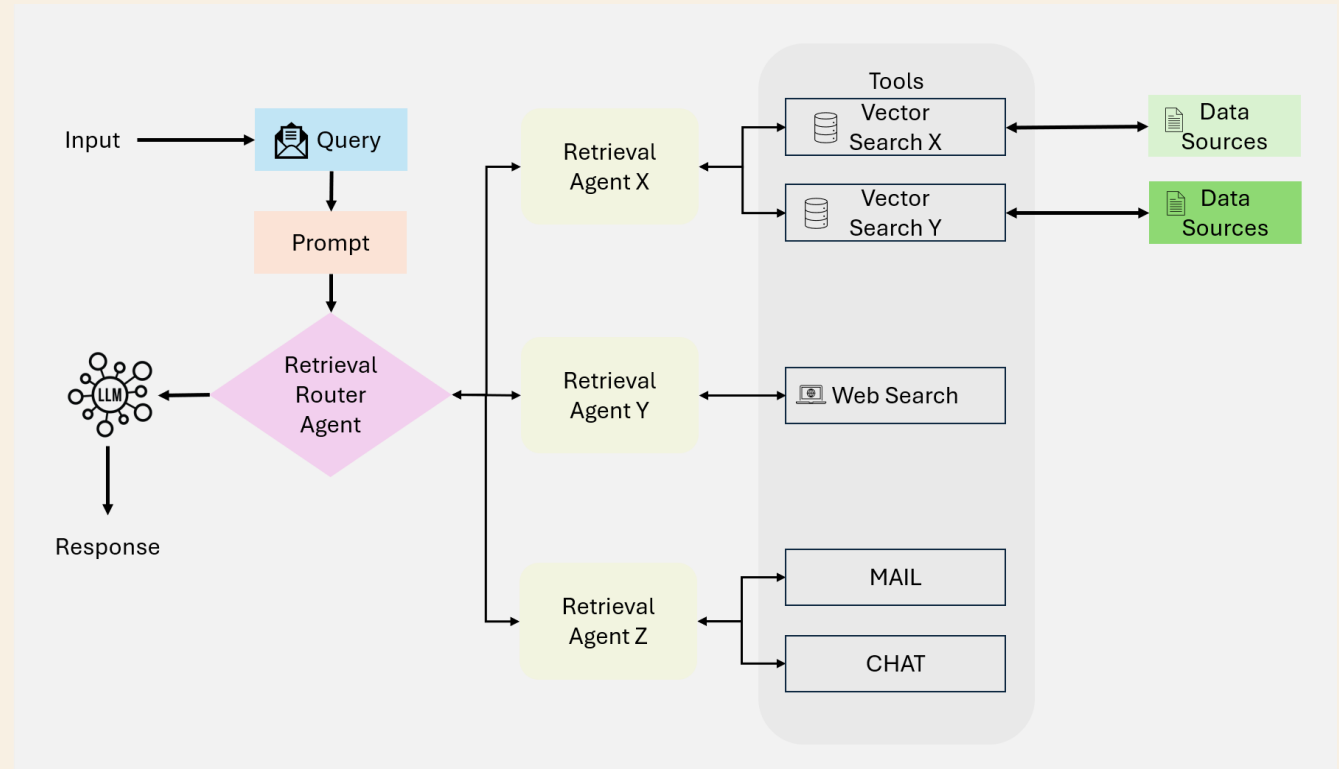
- Agents dynamically divide tasks (e.g., retrieval, reasoning, synthesis).
- Each agent specializes in a specific sub-task.
- Results are aggregated and synthesized into a coherent output.

Advantages:

- Better performance for distributed, multi-step tasks.
- Increased modularity and scalability.

Limitations:

- Coordination complexity increases with the number of agents.
- Risk of redundancy or conflicts between agents.



Hierarchical Agentic RAG

Key Idea: Organizes agents in a **hierarchy** for better task prioritization and delegation.

Workflow:

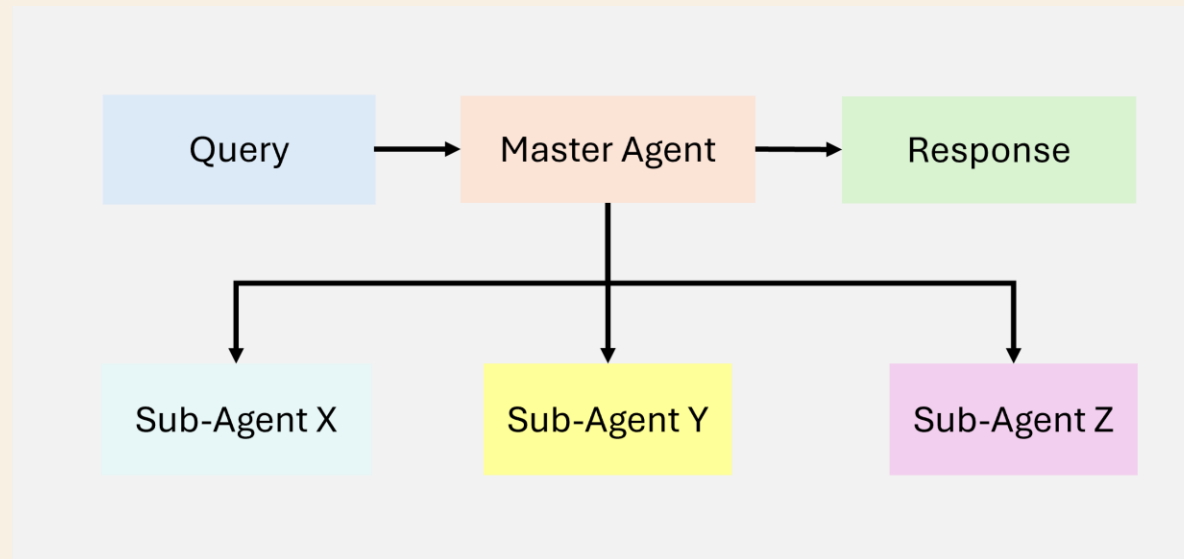
- A top-level agent orchestrates subtasks among lower-level agents.
- Each lower-level agent handles a specific part of the process.
- Results are iteratively refined and integrated at higher levels.

Advantages:

- Scalable for large and complex tasks.
- Modular design facilitates specialization.

Limitations:

- Requires sophisticated orchestration mechanisms.
- Potential bottlenecks at higher levels of the hierarchy.



Corrective Agentic RAG

Key Idea: **Feedback loops** enable agents to evaluate and refine their outputs iteratively.

Workflow:

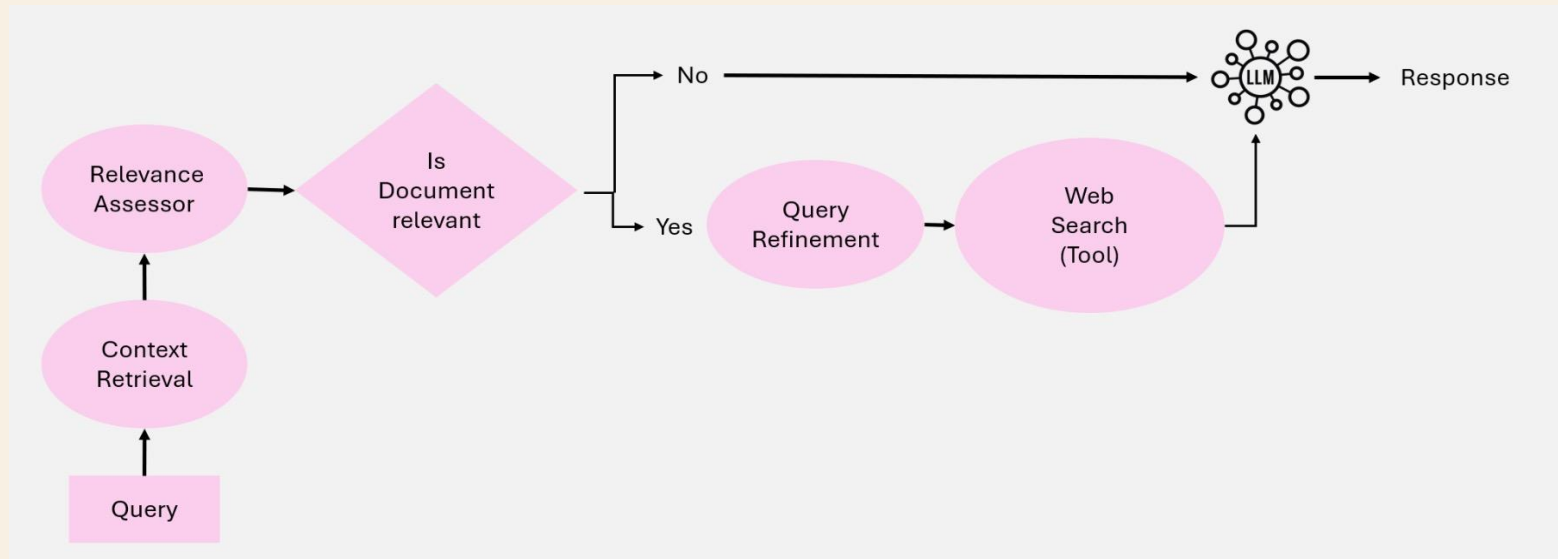
- Initial response is generated by the agent.
- A critic module evaluates the response for errors or inconsistencies.
- The agent refines the response based on feedback.
- Steps 2-3 repeat until the output meets quality standards.

Advantages:

- High accuracy and reliability through iterative improvements.
- Useful for error-prone or high-stakes tasks.

Limitations:

- Increased computational overhead.
- Feedback mechanisms must be well-designed to avoid infinite loops.



Adaptive Agentic RAG

Key Idea: **Dynamically** adjusts retrieval strategies and workflows based on task requirements.

Workflow:

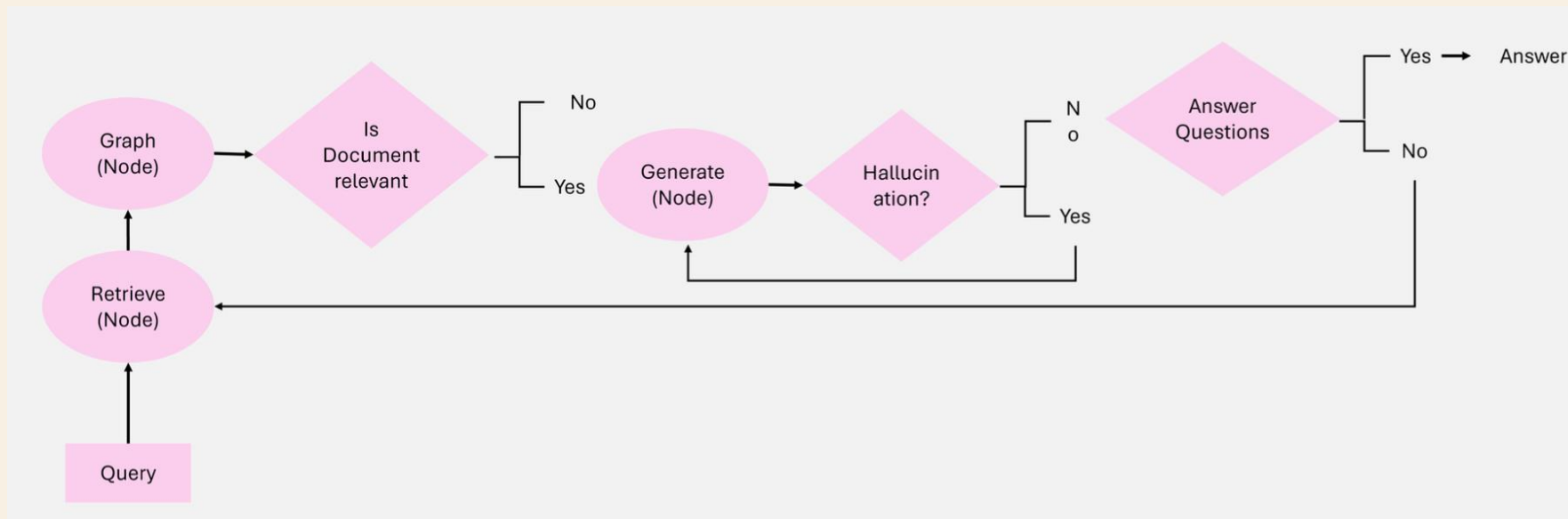
- For straightforward queries, the system avoids unnecessary retrieval, directly leveraging the LLM for response generation.
- For simple queries, it employs a single-step retrieval process to fetch relevant context.
- For complex queries, it activates multi-step retrieval to ensure iterative refinement and enhanced reasoning.

Advantages:

- High flexibility for varied tasks and dynamic environments.
- Improves context relevance and user satisfaction.

Limitations:

- Challenging to design robust adaptation mechanisms.
- Computational overhead for real-time adjustments.



Graph-Based Agentic RAG

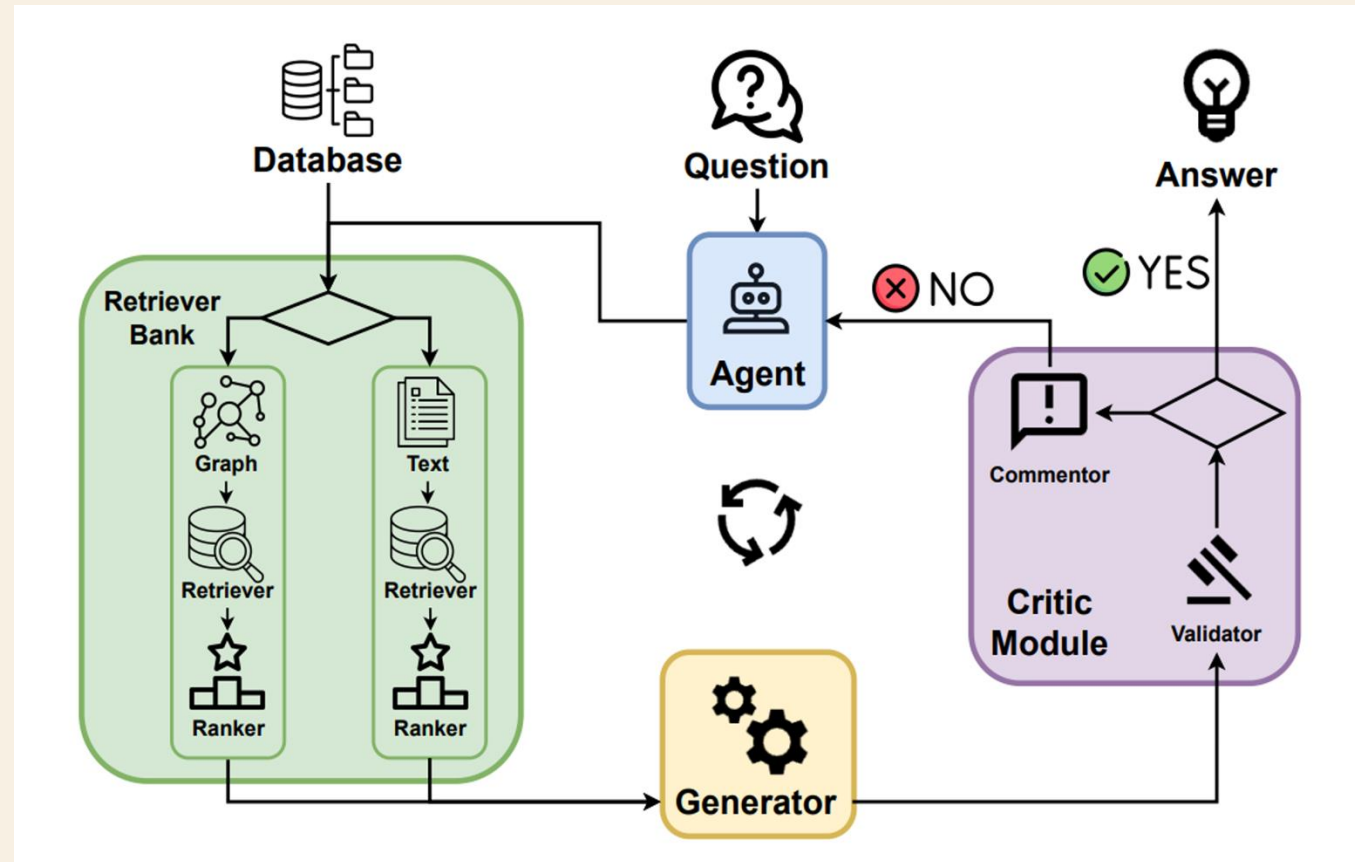
Key Idea: Dynamically assigns tasks to specialized agents using **graph knowledge** bases and feedback loops.

Workflow:

- Extract relationships from graph knowledge bases (e.g., disease-to-symptom mappings).
- Complement with unstructured data from external sources.
- Use a critic module to validate results and iteratively improve.

Advantages:

- Combines structured (graph) and unstructured (text) data.
- Modular and scalable for complex tasks.
- Ensures high accuracy through iterative refinement.



Agenda

- **Taxonomy of Agentic RAG Systems**
- **Applications of Agentic RAG**

Applications of Agentic RAG

- 1. Customer Support and Virtual Assistants**
- 2. Healthcare and Personalized Medicine**
- 3. Legal and Contract Analysis**
- 4. Finance and Risk Analysis**
- 5. Education and Personalized Learning**
- 6. Graph-Enhanced Applications in Multimodal Workflows**

Customer Support and Virtual Assistants

Problem: Providing **contextually relevant** and **dynamic responses** to customer queries.

Use Case: Twitch Ad Sales Enhancement

For instance, Twitch leveraged an agentic workflow with RAG on Amazon Bedrock to streamline ad sales. The system dynamically retrieved advertiser data, historical campaign performance, and audience demographics to generate detailed ad proposals, significantly boosting operational efficiency.

Key Benefits:

- **Improved Response Quality:** Personalized and context-aware replies enhance user engagement.
- **Operational Efficiency:** Reduces the workload on human support agents by automating complex queries.
- **Real-Time Adaptability:** Dynamically integrates evolving data, such as live service outages or pricing updates.

Healthcare and Personalized Medicine

Problem: Rapid retrieval and synthesis of medical knowledge for diagnostics, treatment planning, and research.

Use Case: Patient Case Summary

Agentic RAG systems have been applied in generating patient case summaries. For example, by integrating electronic health records (EHR) and up-to-date medical literature, the system generates comprehensive summaries for clinicians to make faster and more informed decisions.

Key Benefits:

- **Personalized Care:** Tailors recommendations to individual patient needs.
- **Time Efficiency:** Streamlines the retrieval of relevant research, saving valuable time for healthcare providers.
- **Accuracy:** Ensures recommendations are based on the latest evidence and patient-specific parameters.

Legal and Contract Analysis

Problem: Analyzing complex legal documents and extracting actionable insights.

Use Case: Contract Review

A legal agentic RAG system can analyze contracts, extract critical clauses, and identify potential risks. By combining semantic search capabilities with legal knowledge graphs, it **automates** the tedious process of contract review, ensuring compliance and mitigating risks.

Key Benefits:

- **Risk Identification:** Automatically flags clauses that deviate from standard terms.
- **Efficiency:** Reduces the time spent on contract review processes.
- **Scalability:** Handles large volumes of contracts simultaneously.

Finance and Risk Analysis

Problem: Analyzing large-scale financial datasets and identifying trends, risks, and opportunities.

Use Case: Auto Insurance Claims Processing

In auto insurance, Agentic RAG can **automate** claim processing. For example, by retrieving policy details and combining them with accident data, it generates claim recommendations while ensuring compliance with regulatory requirements.

Key Benefits:

- **Real-Time Analytics:** Delivers insights based on live market data.
- **Risk Mitigation:** Identifies potential risks using predictive analysis and multi-step reasoning.
- **Enhanced Decision-Making:** Combines historical and live data for comprehensive strategies.

Education and Personalized Learning

Problem: Delivering **personalized** and adaptive learning experiences for diverse learners.

Use Case: Research Paper Generation

In higher education, Agentic RAG has been used to assist researchers by synthesizing key findings from multiple sources. For instance, a researcher querying, "What are the latest advancements in quantum computing?" receives a concise summary enriched with references, enhancing the quality and efficiency of their work.

Key Benefits:

- **Tailored Learning Paths:** Adapts content to individual student needs and performance levels.
- **Engaging Interactions:** Provides interactive explanations and personalized feedback.
- **Scalability:** Supports large-scale deployments for diverse educational environments.

Graph-Enhanced Applications in Multimodal Workflows

Problem: Tackling tasks requiring **relational understanding** and multi-modal data integration.

Use Case: Market Survey Generation

GEAR enables the synthesis of text, images, and videos for marketing campaigns. For example, querying, “What are the emerging trends in eco-friendly products?” generates a detailed report enriched with customer preferences, competitor analysis, and **multimedia** content.

Key Benefits:

- **Multi-Modal Capabilities:** Integrates text, image, and video data for comprehensive outputs.
- **Enhanced Creativity:** Generates innovative ideas and solutions for marketing and entertainment.
- **Dynamic Adaptability:** Adapts to evolving market trends and customer needs.



Thank you!