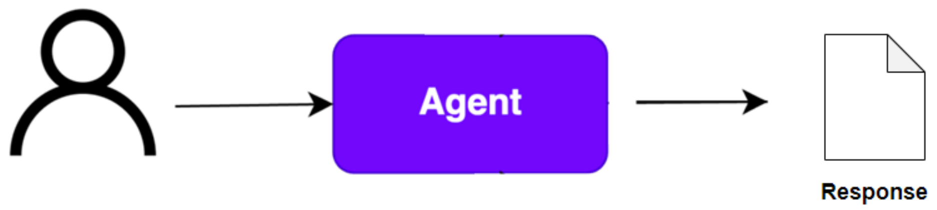
Large Language Models

Agent Architectures

Riccardo Coppola

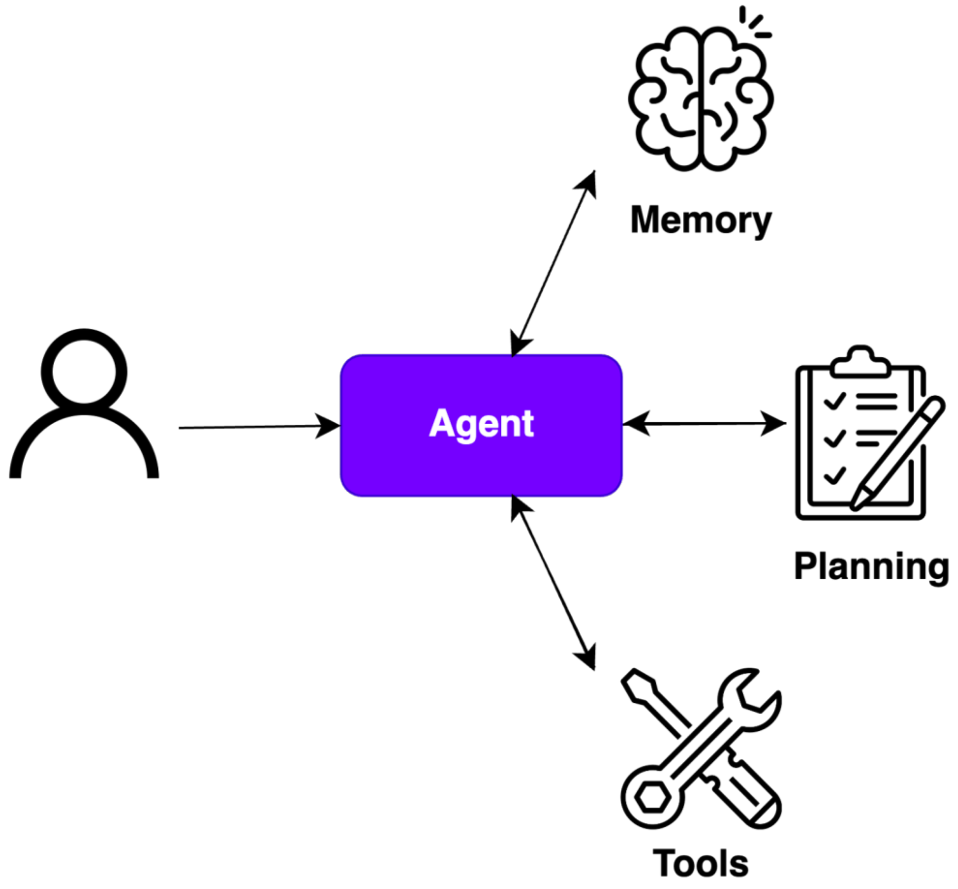
Agentic Behaviour in LLMs

- In **traditional LLM applications**, the model follows a predetermined control flow — a fixed sequence of steps before and after LLM calls.
  - For example, a Retrieval-Augmented Generation (RAG) system retrieves relevant documents in response to a question and passes them to the LLM to ground its response. While effective, this approach limits the model to a static workflow, potentially restricting its problem-solving capabilities.



Agentic Behaviour in LLMs

- **Agentic behavior** refers to empowering an LLM to decide its own control flow to solve more complex problems. In other words, the model becomes an "agent" that can make decisions about which steps to take, which tools to use and when to terminate a process. This flexibility allows for more sophisticated applications, enabling the LLM to handle tasks that require dynamic decision-making and adaptability.



Why agentic concepts matter

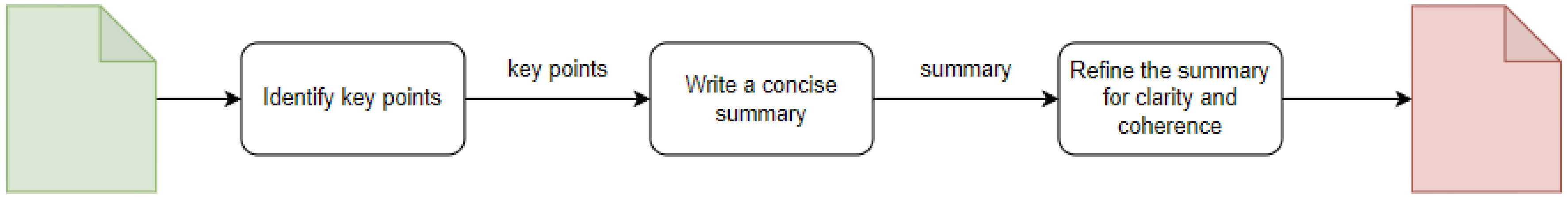
- Introducing agentic behavior into LLM workflows has lots of advantages:
  - **Enhanced Problem-Solving**: Agents can tackle more complex tasks by choosing the most appropriate actions.
  - **Dynamic Control Flow**: Agents are not limited to a fixed sequence of steps, allowing for more flexible and efficient processes.
  - **Improved Efficiency**: By deciding when enough information has been gathered, agents can reduce unnecessary computations.

LLM Chains

- An LLM Chain is a series of prompts and operations that guide an LLM through a sequence of tasks, enabling more complex and nuanced AI interactions.
  - Think of it as a recipe for AI: just as a chef follows a series of steps to create a gourmet dish, an LLM Chain provides a structured workflow for an AI to accomplish sophisticated tasks.

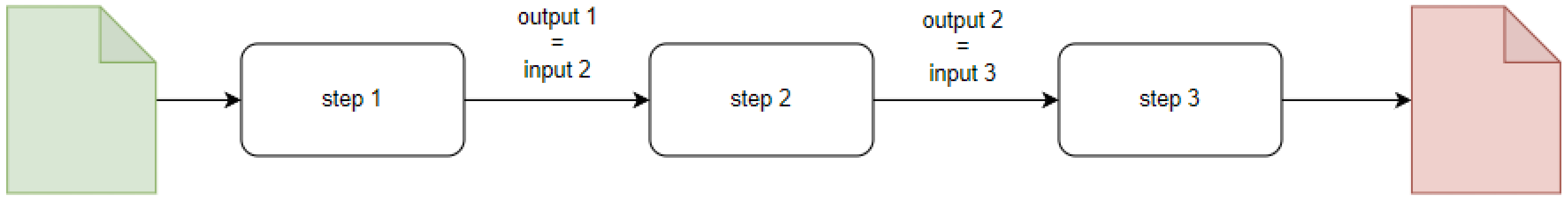
LLM Chains

- For example, a simple LLM Chain for summarizing a long article might involve the following steps:
  1. Prompt the LLM to read the article
  2. Ask it to identify key points
  3. Request a concise summary based on those points
  4. Refine the summary for clarity and coherence



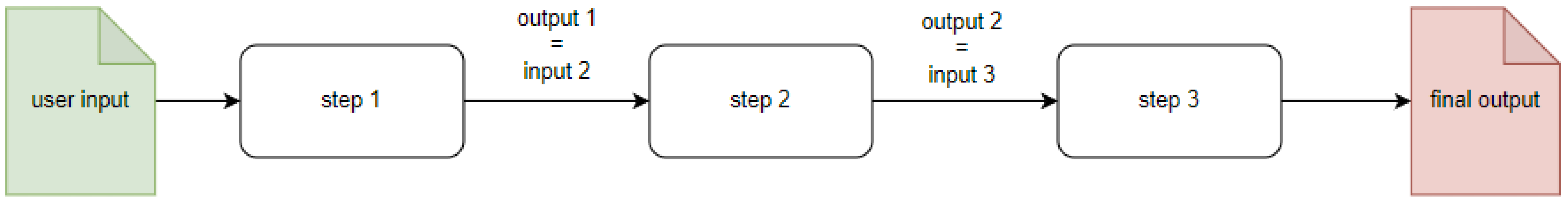
Sequential chain

- In general, such example is a **sequential chain**
- Sequential chains, in their simplest form, consist of steps where each step takes one input and produces one output. The output from one step becomes the input for the next.



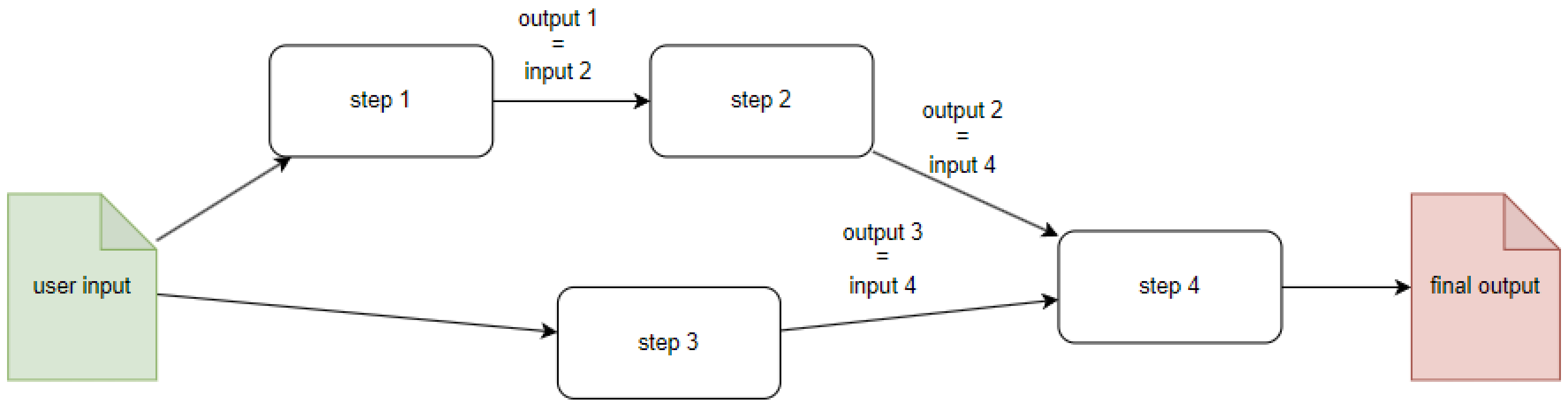
Sequential chain

- This straightforward approach is effective when dealing with sub-chains designed for singular inputs and outputs. It ensures a smooth and continuous flow of information, with each step seamlessly passing its output to the subsequent step.



Tree Chain

- Not all of the sequential chains operate with a single string input and output. In more intricate setups, these chains handle multiple inputs and generate multiple final outputs.



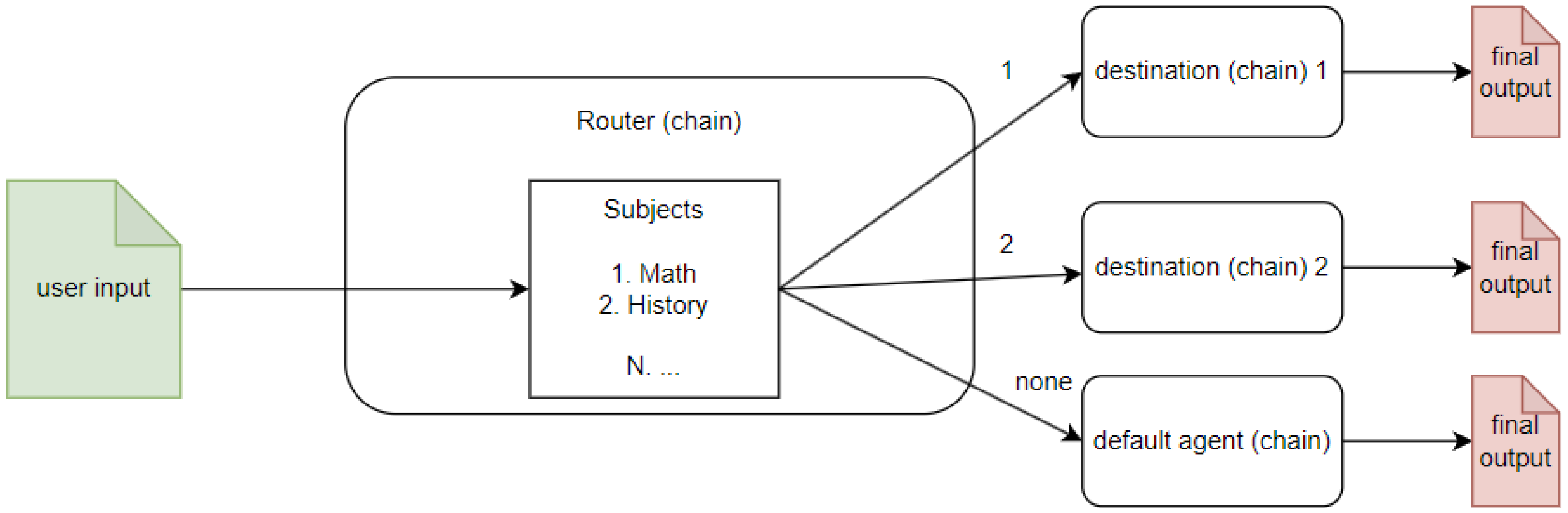
Router Chain

- The Router Chain is used for complicated tasks. If we have multiple subchains, each of which is specialized for a particular type of input, we could have a router chain that decides which subchain to pass the input to.

- It consists of:
  - **Router Chain:** It is responsible for selecting the next chain to call.
  - **Destination Chains:** Chains that the router chain can route to.
  - **Default chain:** Used when the router can’t decide which subchain to use.

Router Chain

- Therouter chain acts as a decision-maker, determining which specialized subchain to send the input to. Essentially, it enables the seamless routing of inputs to the appropriate subchains, ensuring efficient and precise processing based on the input's specific characteristics.



From chains to agents

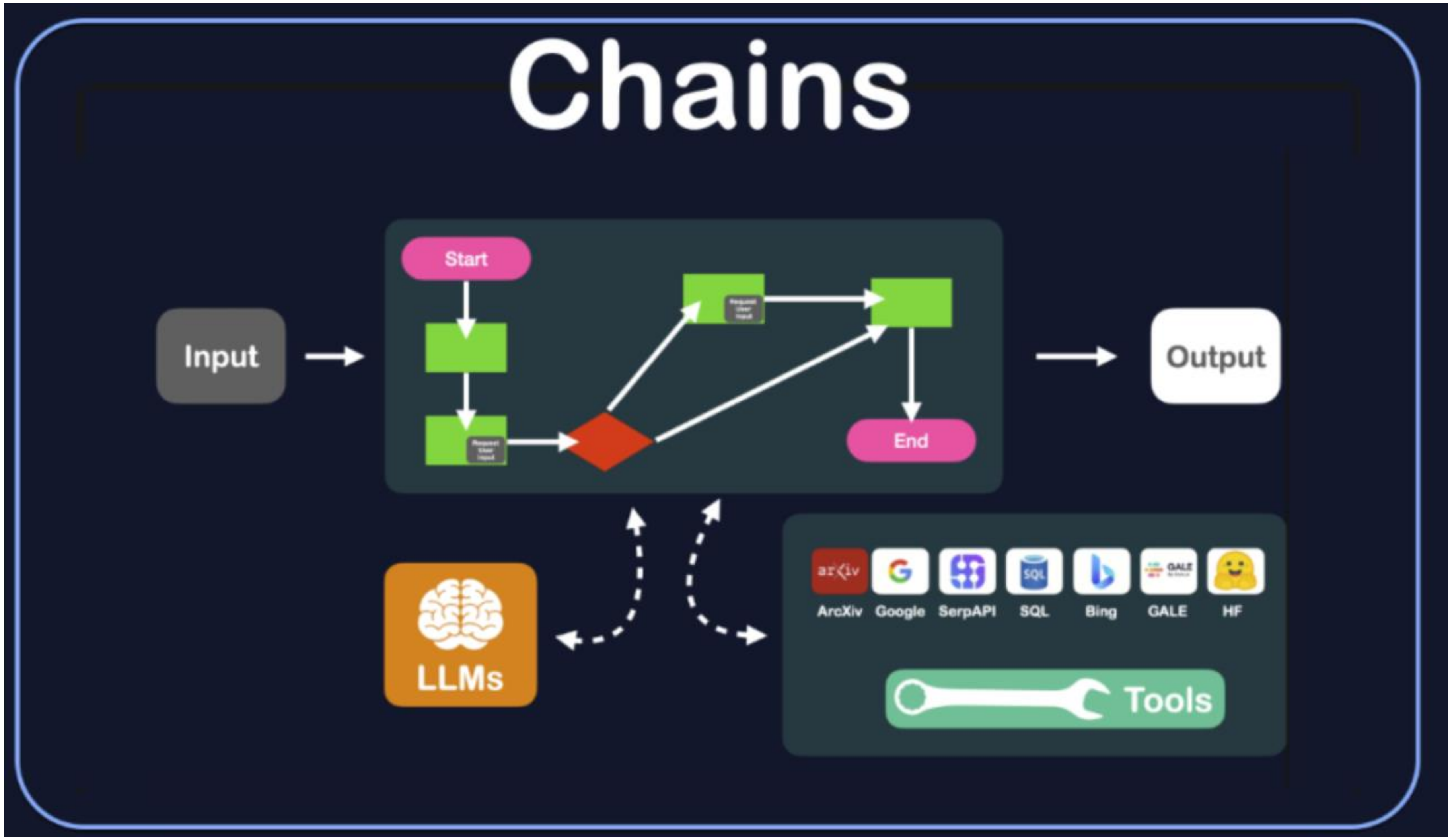
- An agentic AI system is capable of setting its own goals, planning actions to achieve those goals, and making independent decisions based on its understanding of the environment and task at hand. Unlike reactive AI, which simply responds to inputs, agentic AI can take initiative, adapt to changing circumstances, and even learn from its experiences.

- LLM Chains serve as the building blocks for agentic behavior, allowing AI to break down complex tasks, reason through multi-step problems, and interact with its environment in more nuanced ways.

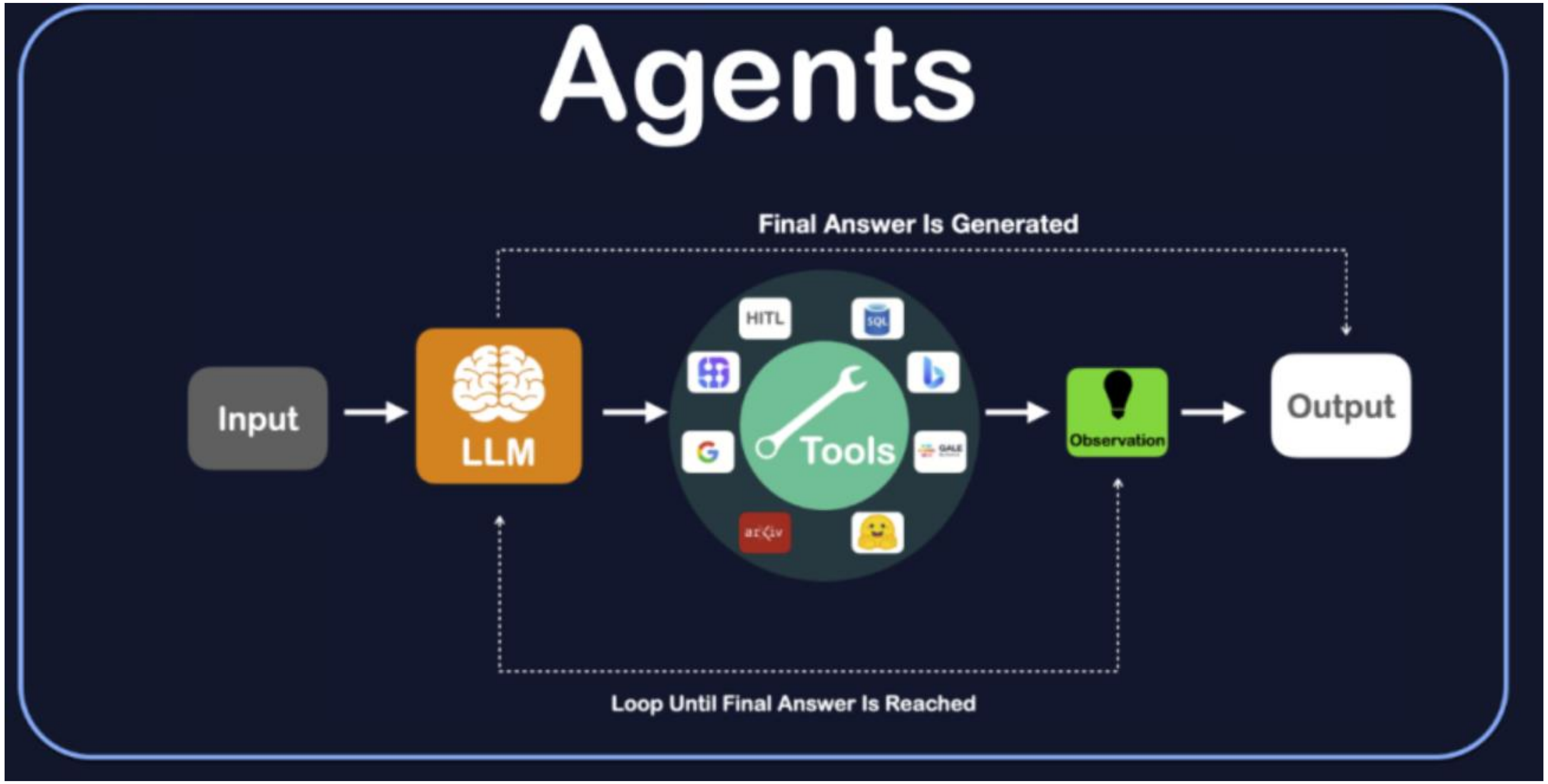
From chains to agents

- To illustrate the difference, consider a virtual assistant:
  - A reactive AI assistant might respond to the query "What's the weather like?" with a current weather report.
  - An agentic AI assistant, upon hearing the same query, might not only provide the weather report but also suggest appropriate clothing for the day, recommend indoor or outdoor activities based on the forecast, and even offer to set reminders for weather-dependent tasks.

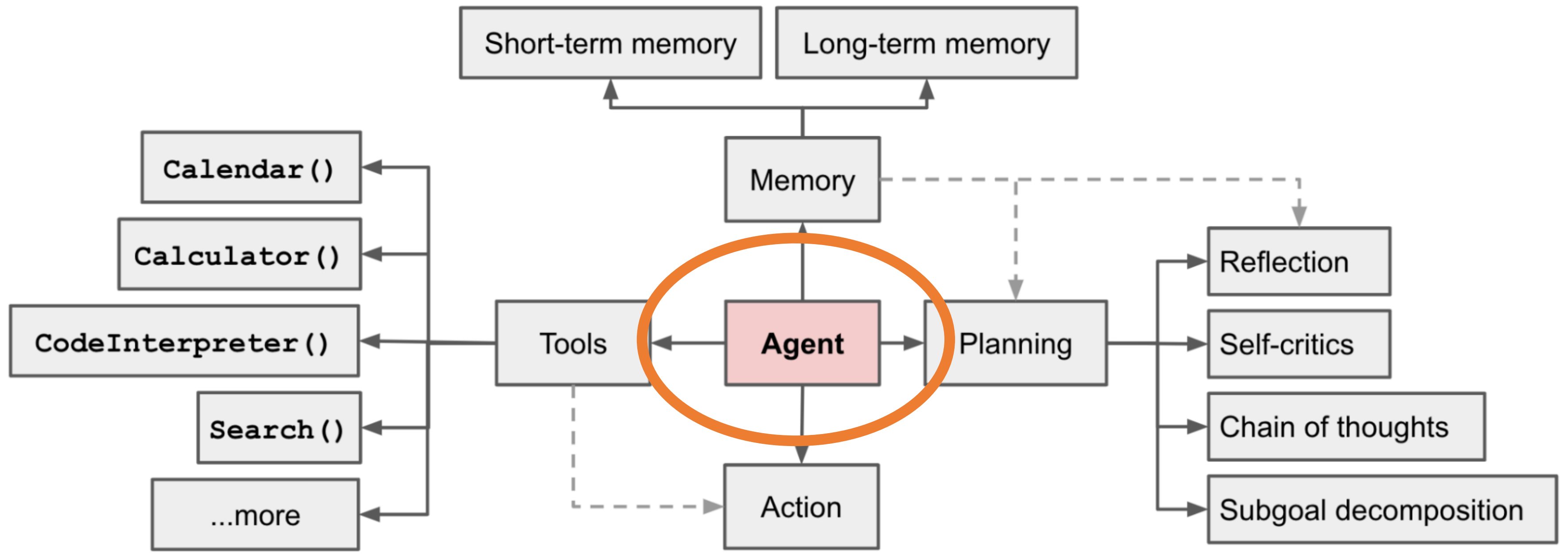
From chains to agents



From chains to agents



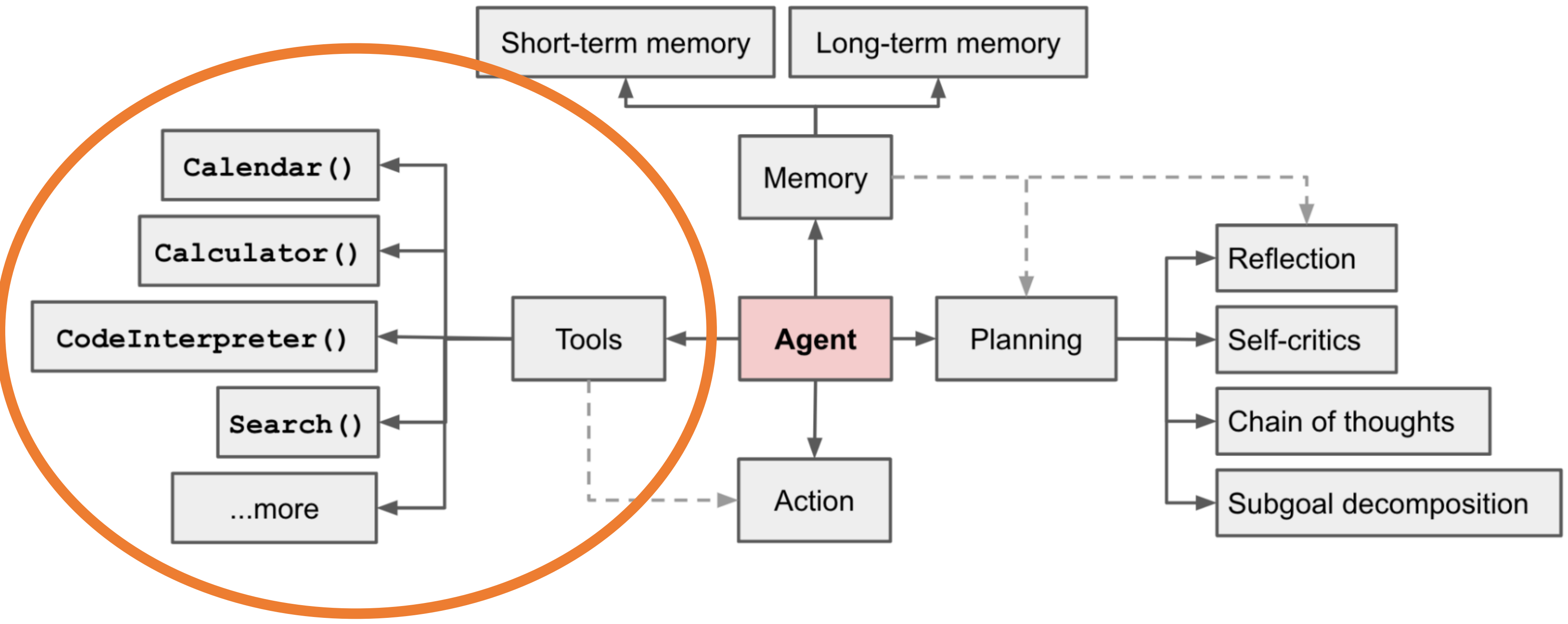
Agentic Architecture



Agent

The **Agent** is the central decision-making entity that orchestrates the use of tools, memory, and planning. It determines how to tackle a given task by selecting the most suitable tool or planning strategy based on the problem's complexity.

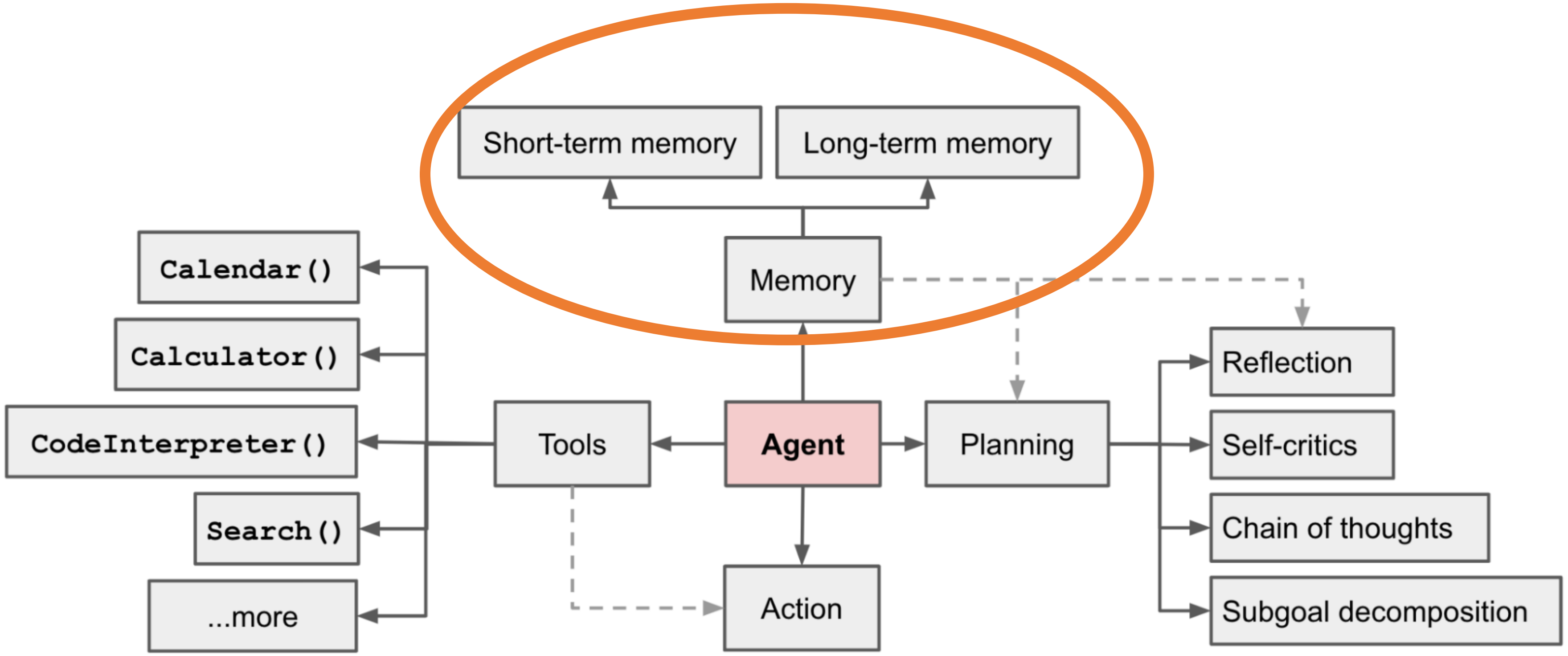
Agentic Architecture



Tools

The **Tools** are specialized extensions that expand the agent's capabilities. Examples include the **Calendar**, which helps schedule events or track deadlines; the **Calculator**, used for precise mathematical operations; the **CodeInterpreter**, enabling coding or data analysis tasks; and the **Search** tool, which fetches real-time information from external sources like the internet. For example, if tasked with planning a project timeline, the agent might use the Calendar to allocate dates and the Calculator to optimize resource allocation.

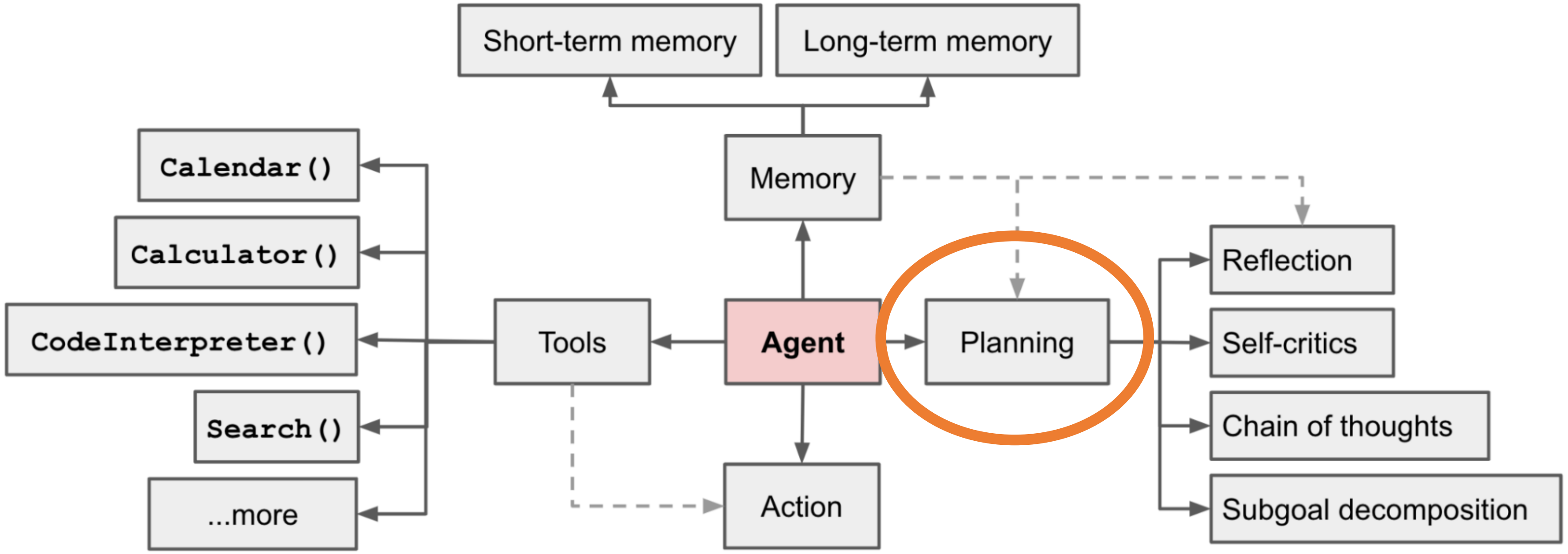
Agentic Architecture



Memory

Memory is divided into **short-term memory** and **long-term memory**. Short-term memory retains temporary information, such as recent instructions or intermediate results, while long-term memory stores persistent knowledge for future reference. For instance, the agent might use short-term memory to temporarily store search results or a conversation context and long-term memory to store a user's preferences, allowing for personalized assistance over time.

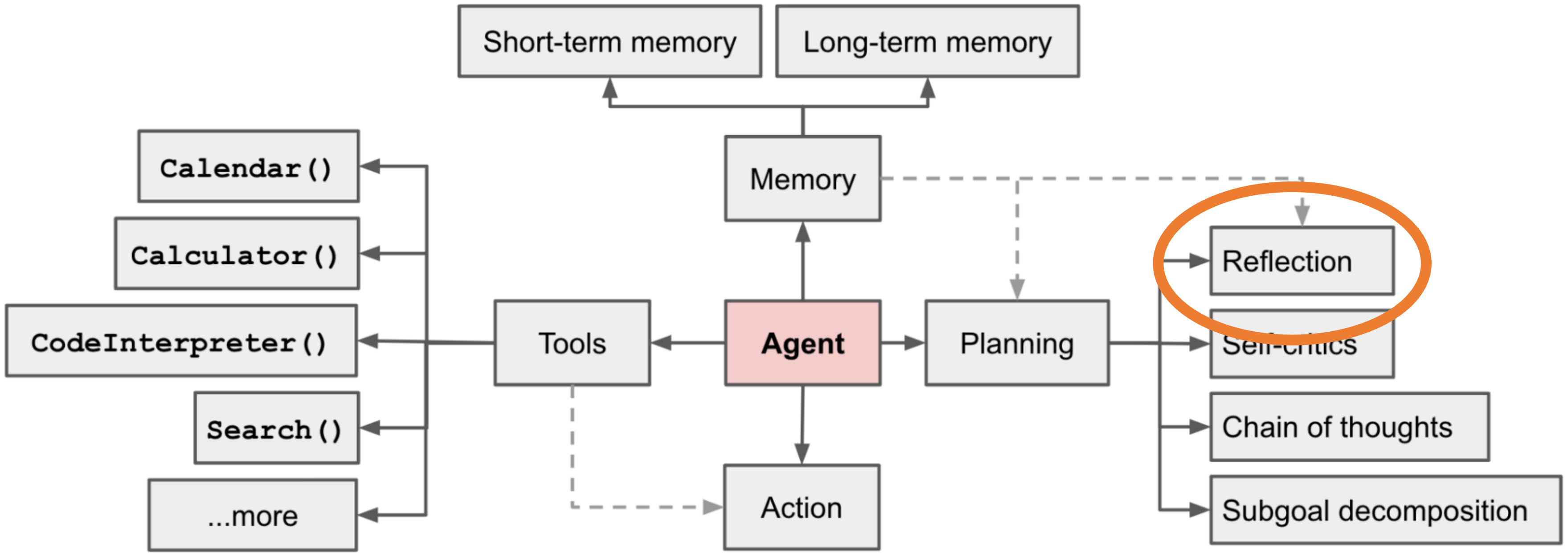
Agentic Architecture



Planning

The **Planning** module allows the agent to devise strategies to solve problems. This involves several the use or combination of several different strategies.

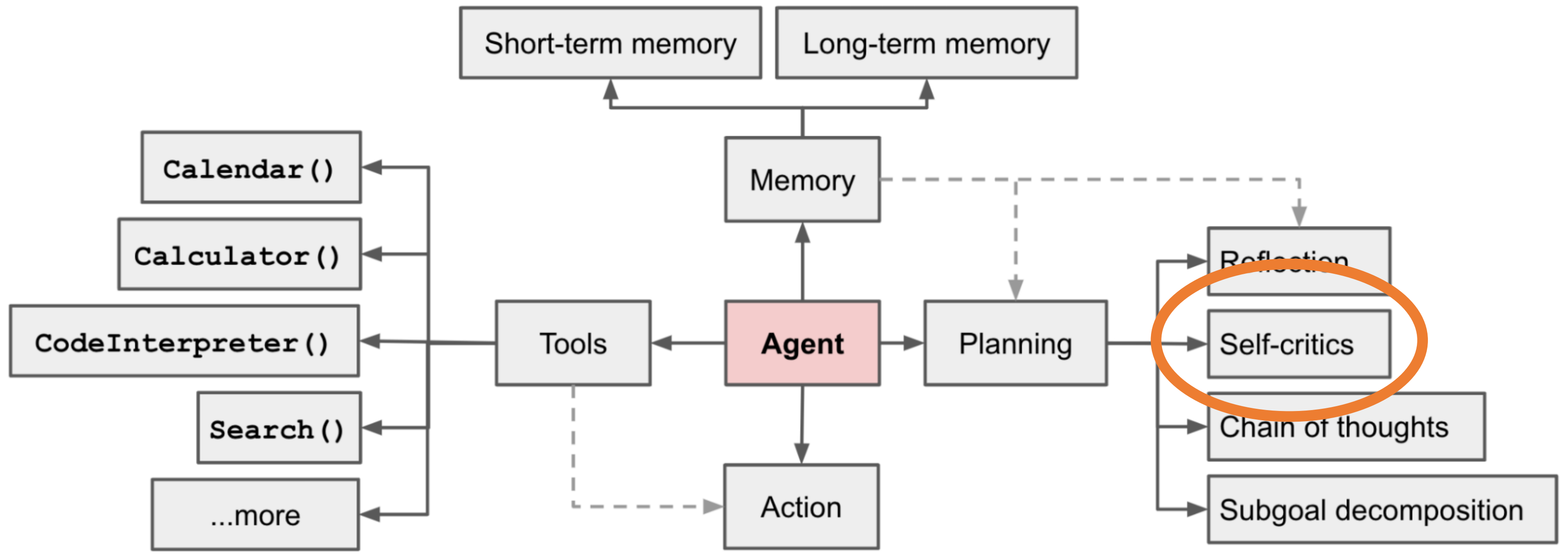
Agentic Architecture



Reflection

Reflection is a meta-cognitive process that allows the agent to evaluate its past decisions and actions to identify areas for improvement. This capability ensures that the agent learns from its successes and failures.

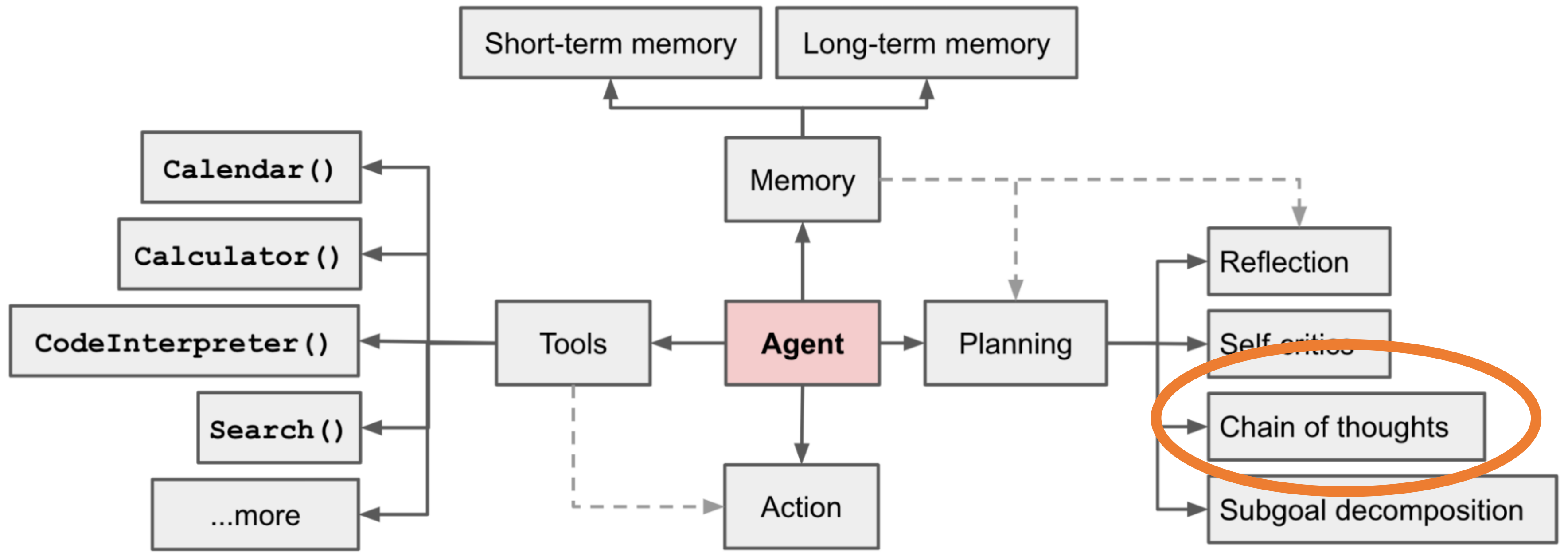
Agentic Architecture



Self-critics

Self-critics enables the agent to analyze its own performance critically and suggest refinements. It operates as an internal feedback mechanism, helping the agent improve its reasoning and outputs.

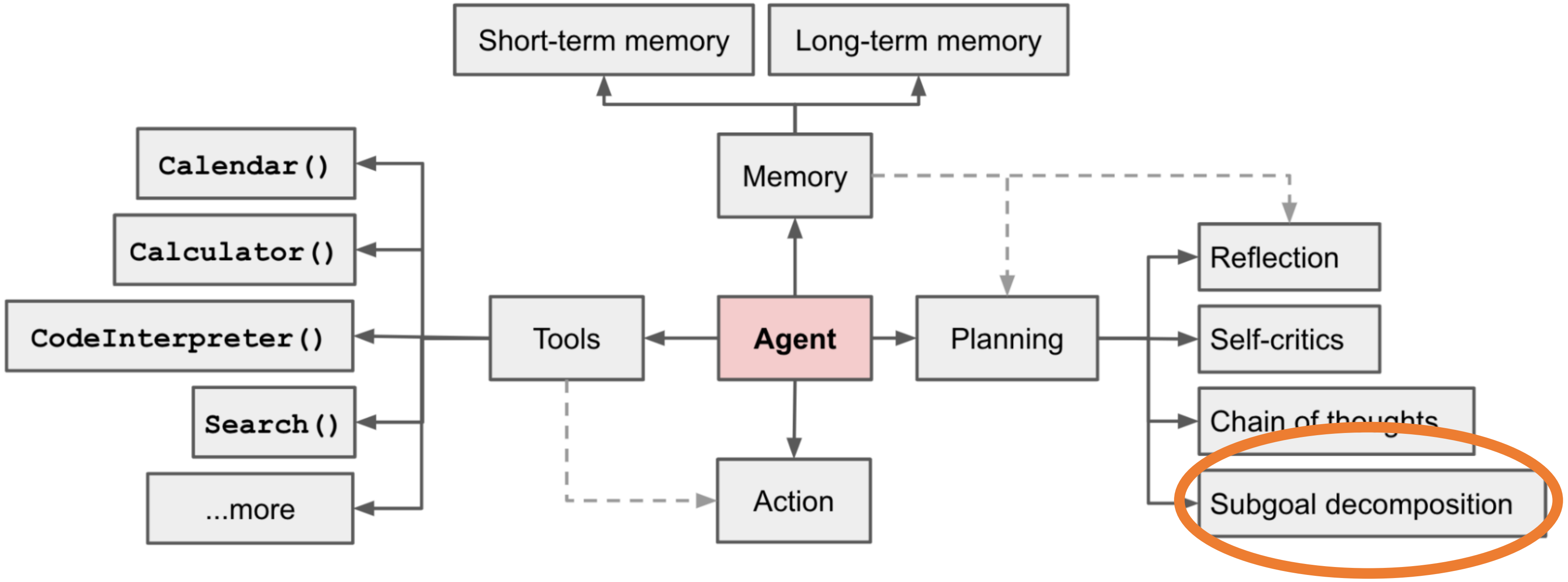
Agentic Architecture



Chain-of-Thoughts

Chain of thoughts involves sequential reasoning to tackle complex, multi-step problems. The agent progresses logically, step by step, ensuring clarity and coherence in problem-solving. This approach minimizes errors and makes the problem-solving process transparent, facilitating debugging and iterative refinement.

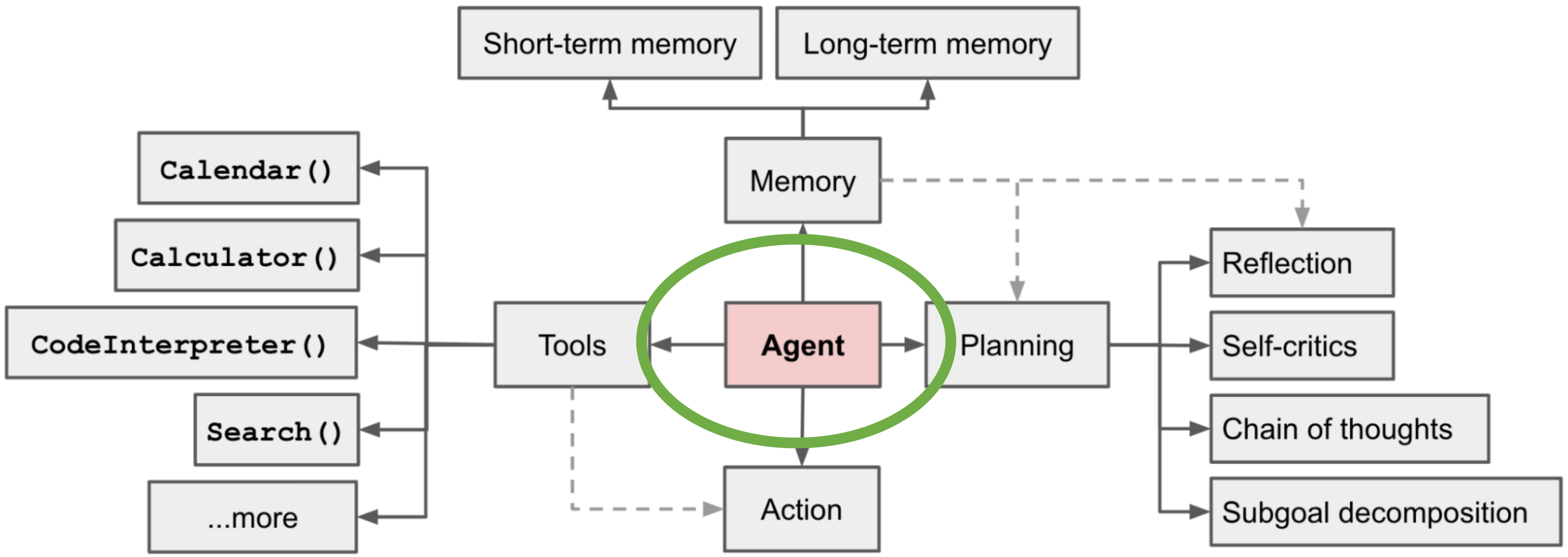
Agentic Architecture



Subgoal decomposition

Subgoal decomposition is the process of breaking down a large, complex problem into smaller, more manageable tasks or milestones. This strategy enables the agent to approach challenges in an organized manner, ensuring steady progress toward the overall objective. This modular approach ensures focus, reduces overwhelm, and enables flexible adjustments if issues arise in individual subgoals.

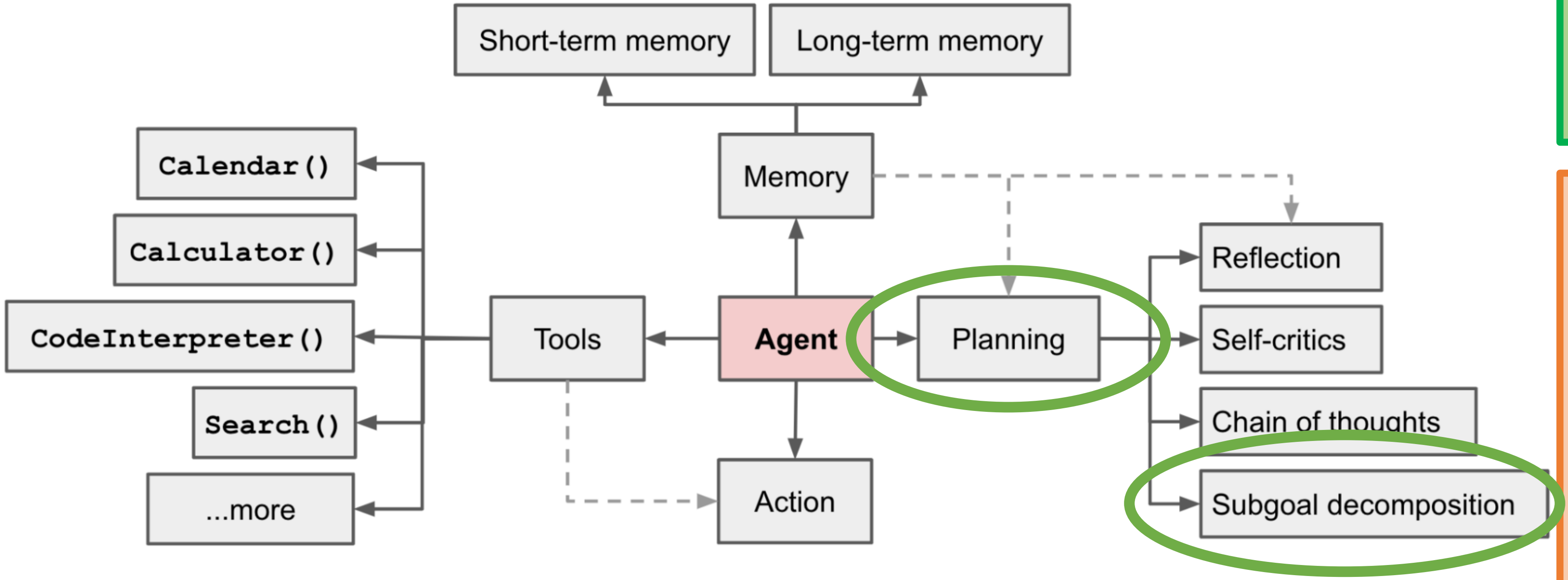
Agentic Architecture: an example



"Plan a two-day workshop for 50 participants on AI and robotics. Include scheduling, budgeting, booking venues, and preparing materials. Ensure efficiency and avoid repeating past mistakes."

At this stage, the **Agent** determines the big picture: which tools to use, whether memory or planning is required, and how to approach the task. It breaks down the goal into smaller **components**.
This will be a prompt for the Planning.

Agentic Architecture: an example



"Break the workshop task into smaller subgoals, such as finding a venue, creating a schedule, estimating the budget, and sending invitations."

Output Example:

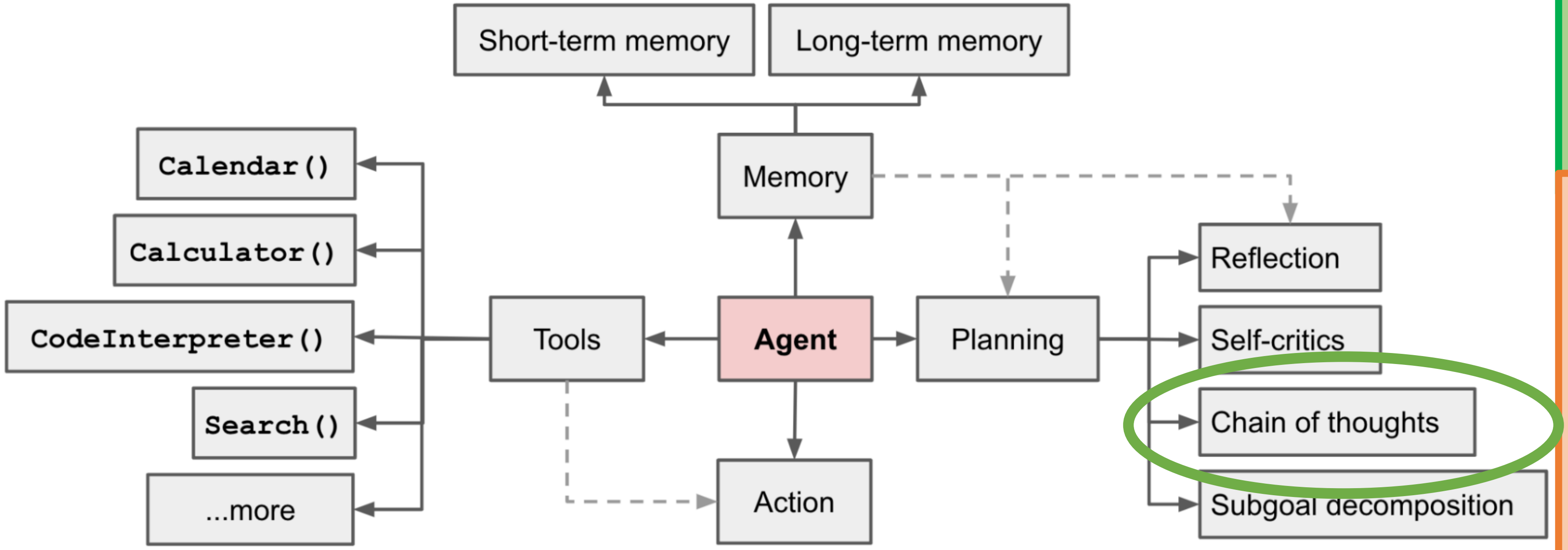
•Subgoal 1: Search for and book a venue.

•Subgoal 2: Draft a schedule with time slots for presentations, breaks, and networking.

•Subgoal 3: Calculate the budget, including food, venue, and materials.

•Subgoal 4: Prepare and distribute invitations.

Agentic Architecture: an example



"For subgoal 1 (finding a venue), reason step-by-step. Start by searching venues, shortlist based on capacity and location, and then compare costs before finalizing."

Output Example:

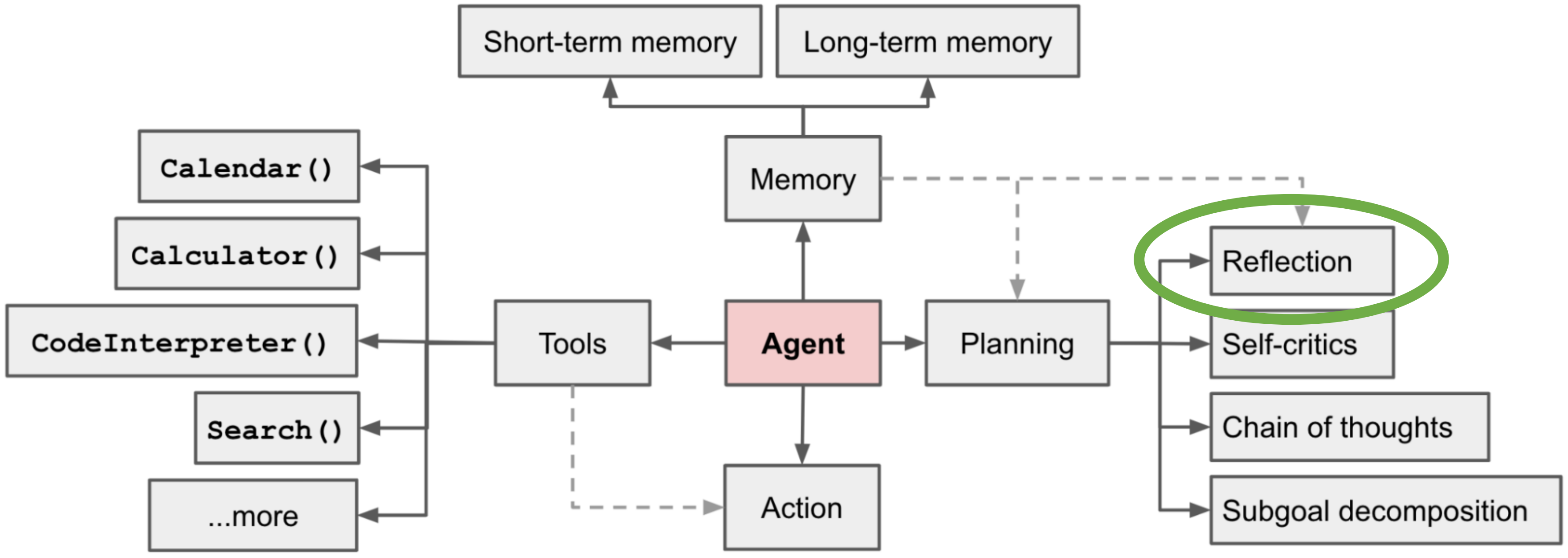
•Step 1: Search for venues that accommodate 50 participants in City X.

•Step 2: Filter results by budget and accessibility.

•Step 3: Contact top 3 venues for availability.

•Step 4: Select the most cost-effective option and book.

Agentic Architecture: an example

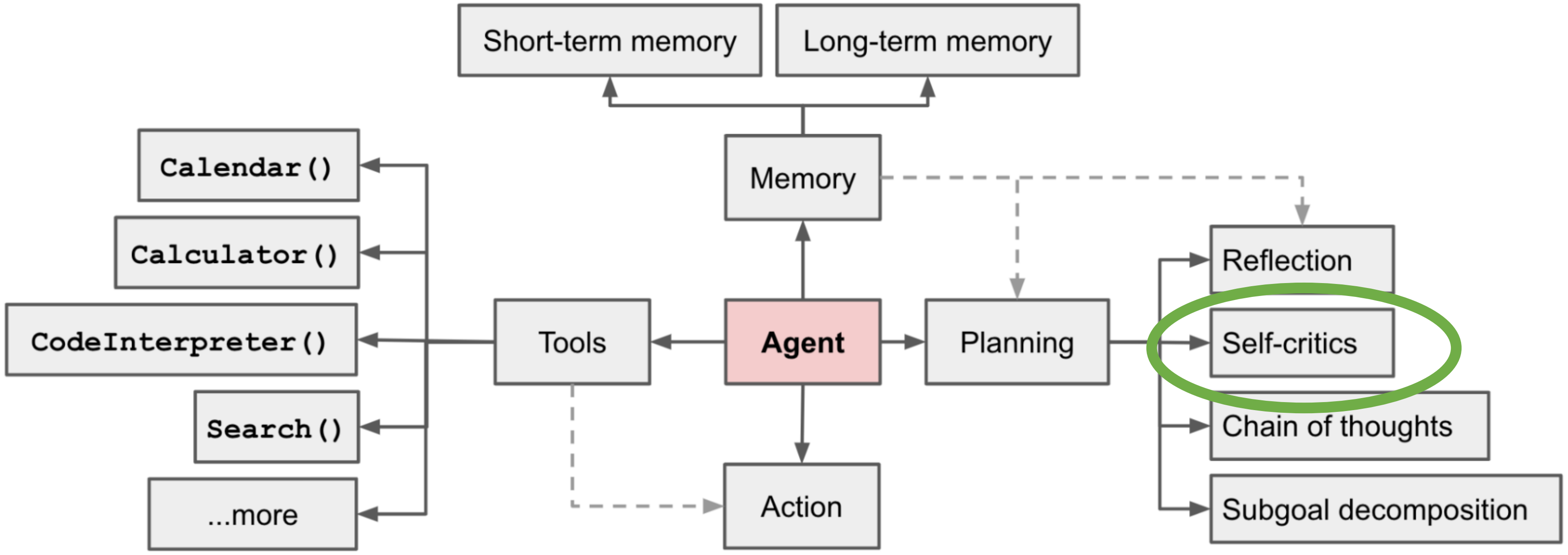


Once a subgoal is completed…

"Now that the venue is booked, reflect on this decision. Was it cost-effective? Did it meet all requirements? If not, what could have been improved?"

Output Example:
"The venue meets capacity and location requirements but costs slightly more than budgeted. In the future, I will consider contacting venues earlier for better deals."

Agentic Architecture: an example

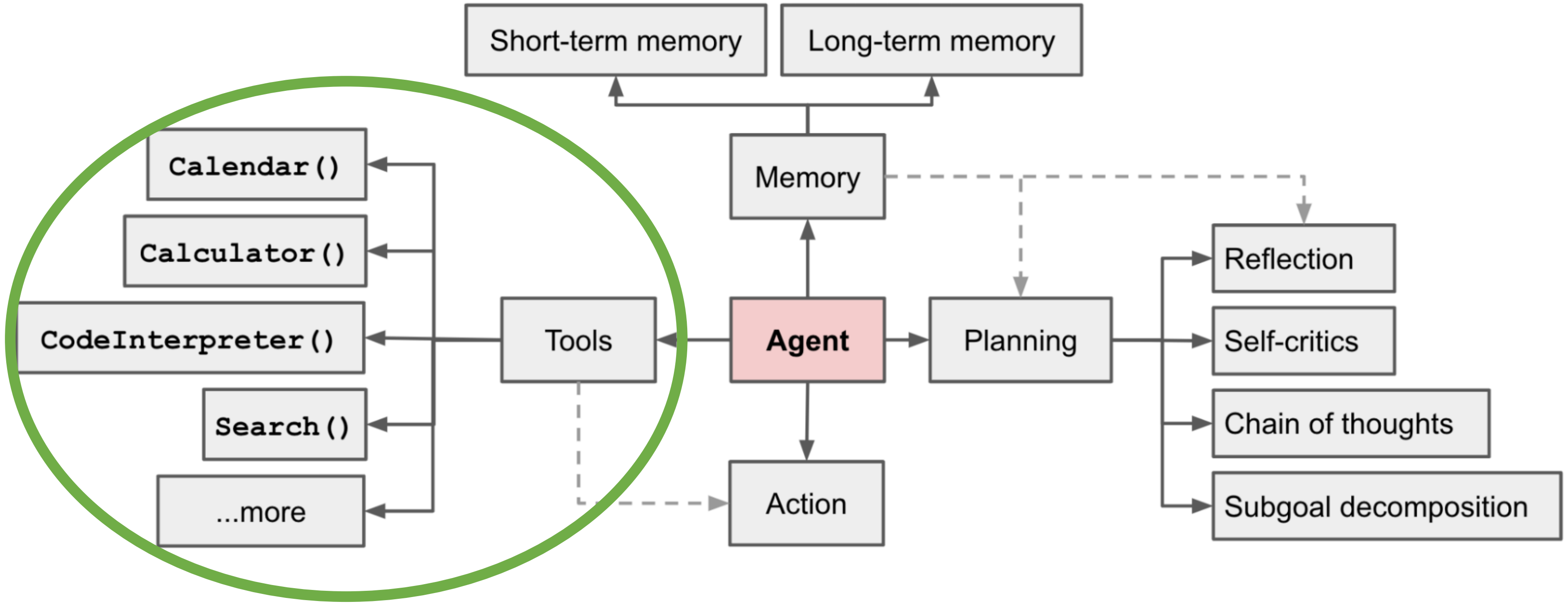


After every subgoal is completed

"Critique the current progress. Are there errors or inefficiencies in the subgoals? Should any priorities or strategies be adjusted?"

Output Example:
"The scheduling plan allocated too much time for lectures, leaving little room for interaction. I recommend reducing lecture times by 15 minutes and adding a Q&A session after each presentation."

Agentic Architecture: an example

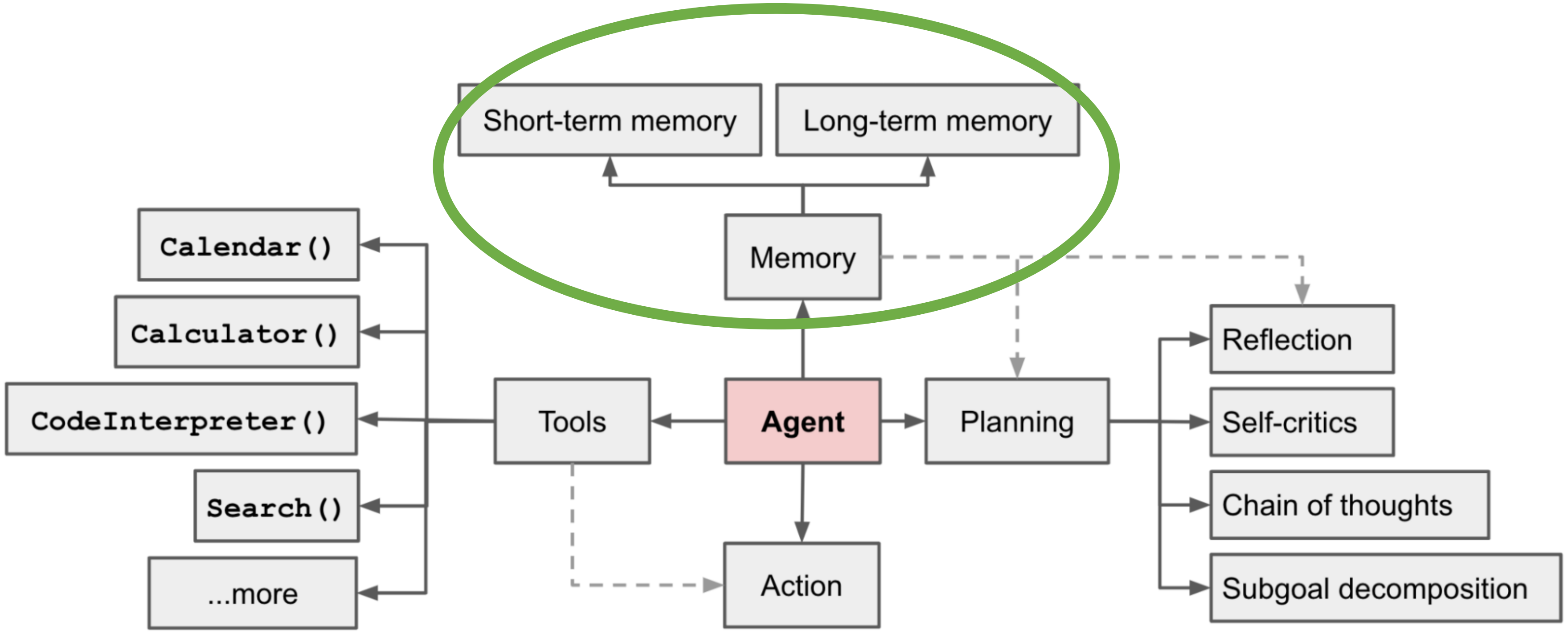


Throughout the process, the **Agent** dynamically uses tools to achieve specific tasks.

Tools may or may not be LLM applications.

- **Calendar**: "Schedule sessions for two days, ensuring balanced timing for presentations, networking, and breaks."
- **Calculator**: "Estimate the budget for 50 participants, including venue, catering, and materials."
- **Search**: "Find AI experts to invite as speakers."

Agentic Architecture: an example



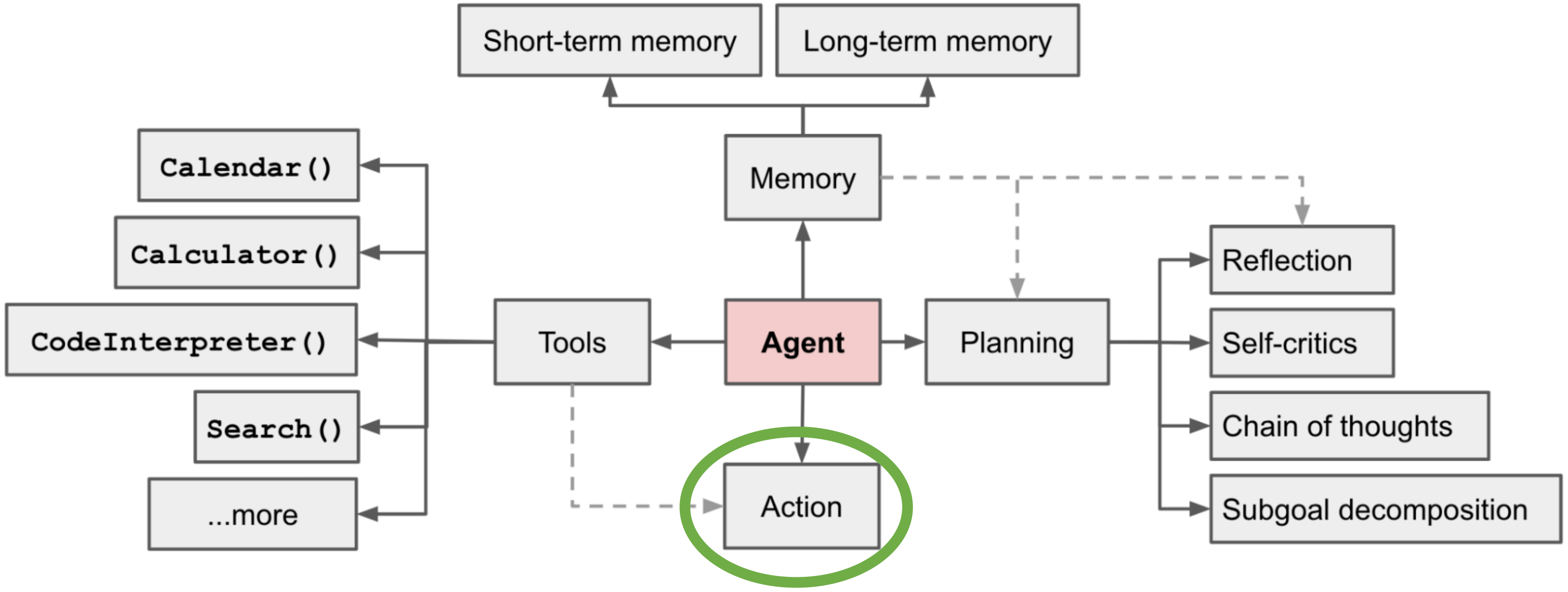
"Store the current workshop schedule in memory for future reference. Retrieve feedback from last year's workshop on the event timing and use it to refine this plan."

Use **Memory** to track progress or retrieve past knowledge.

•**Short-term memory**: Stores temporary information, like shortlisted venues or intermediate calculations.

•**Long-term memory**: Retrieves lessons learned from previous workshops, such as participant feedback about food quality or session timing.

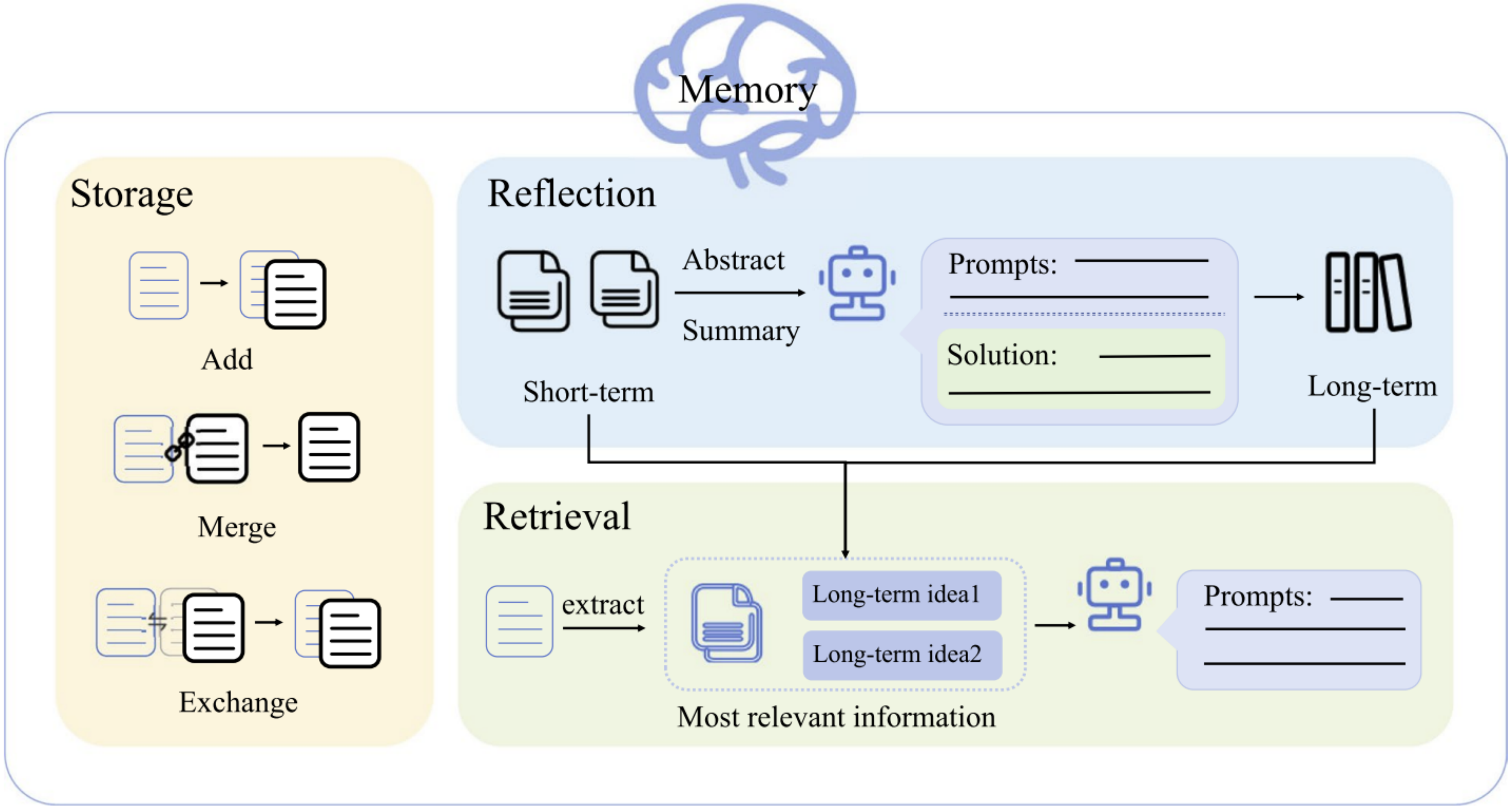
Agentic Architecture: an example



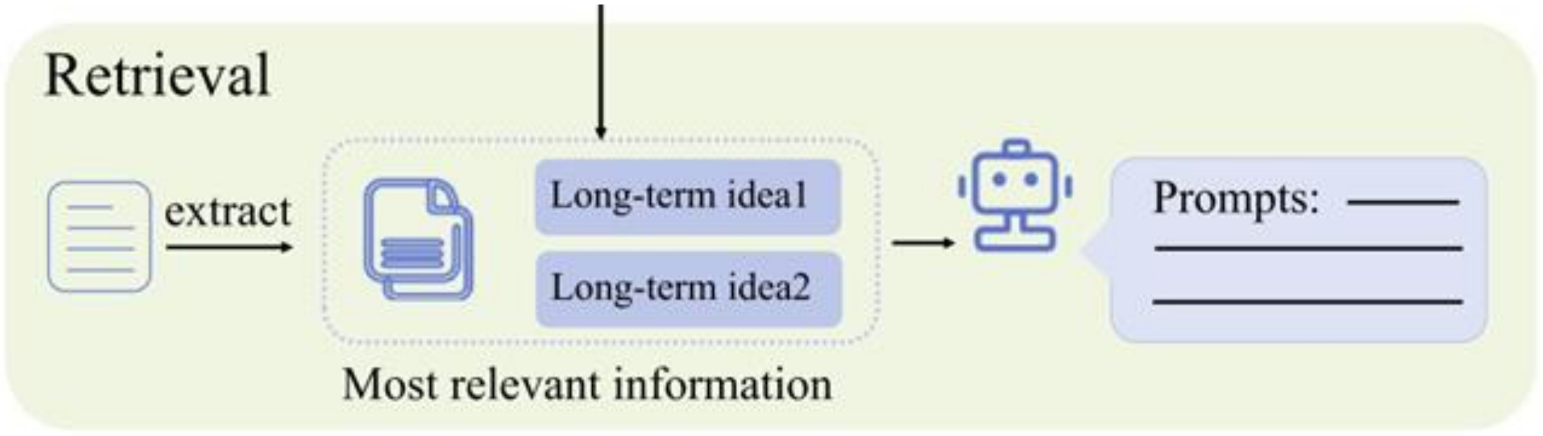
"Book the venue, send invitations, confirm the schedule, and prepare workshop materials as per the finalized plan."

Once everything is planned, the **Agent** executes the final **Action** based on the refined plan.

The memory model



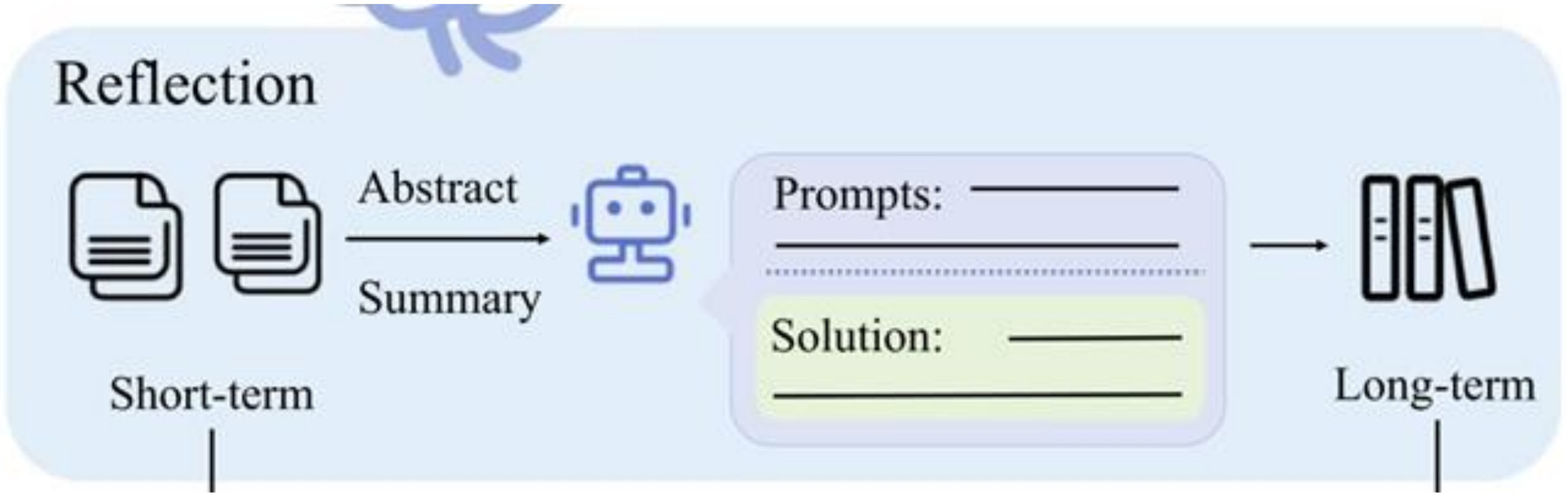
The memory model



Memory retrieval. Memory retrieval aims to enhance decision-making accuracy by extracting valuable information pertinent to the current situation from an agent's memory. This information encompasses various elements such as environmental perception, records of historical interactions, experiential data, and external knowledge.

In scenarios involving short-term memory, the retrieval module typically extracts the entire body of information as content. However, when dealing with long-term memory, the retrieval module generally employs filtering mechanisms to discern and present only the most relevant memories to the agent.

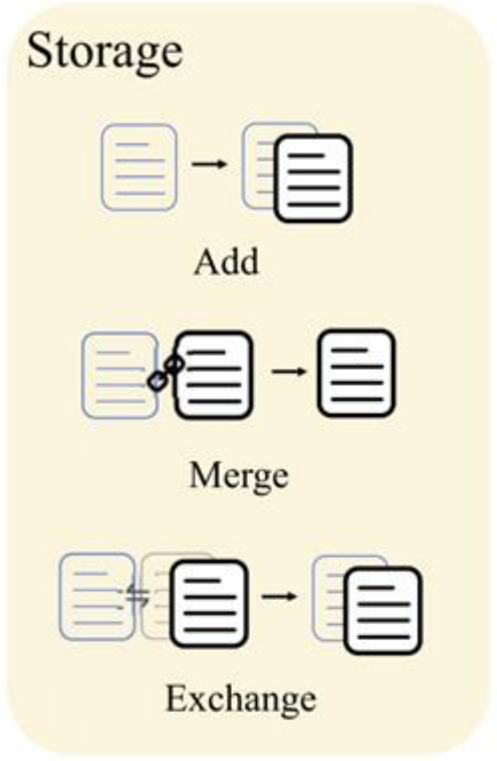
The memory model



Memory reflection. Memory Refection is the process through which agents engage in self-improvement based on the perceived information and learned experience from historical interactions stored in memory. process emulates the human practice of summarizing, refining, and reflecting upon existing knowledge, with the objective of enhancing the agent’s adaptability to new environments and tasks.

The memory refection process typically occurs automatically, with agents independently updating their memory based on newly acquired knowledge, thereby achieving self-recognition updates. In a multiagent environment, a central LLM-based agent exerts control over the memory refection of individual agents.

The memory model



Memory storage. The storage is typically realized through the use of natural language formed text, although it also encompasses multi-modal information such as visual and audio data. The storage format is determined by the specific nature of the task and the attributes of the data modality. By tailoring the storage format to the modality and task requirements, agents can more effectively utilize stored information, thereby enhancing their performance in diverse and complex environments.

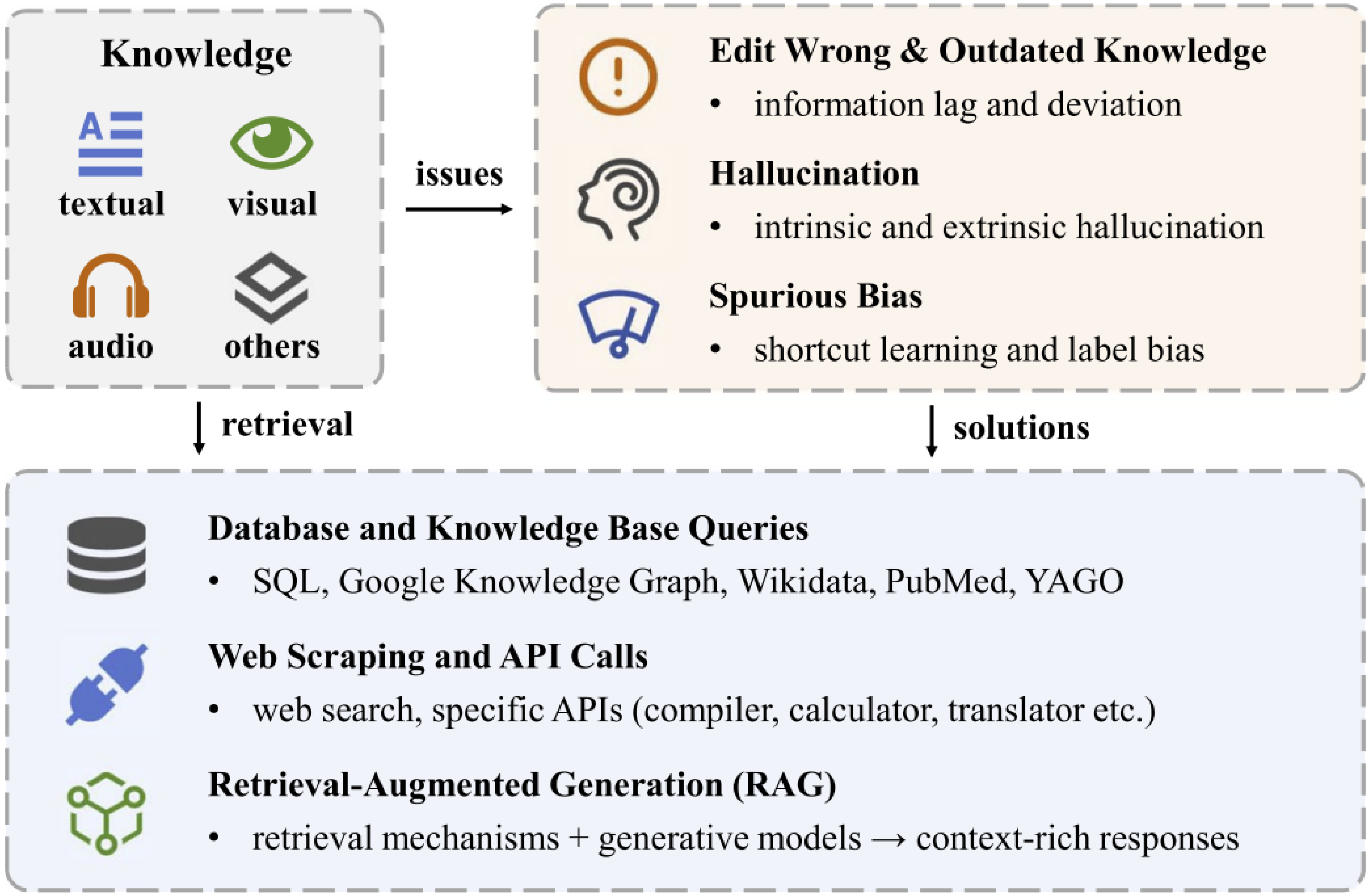
Memory Modifcation. When considering the similarity between new information and existing memories, it is crucial to determine the appropriate method of incorporation: whether to add new information, merge it with existing data, or substitute erroneous existing information.

Utilization of Knowledge

- **Knowledge utilization**. Knowledge utilization focuses on integrating external knowledge (excluding memory information) into LLM based planning.

- By leveraging up-to-date textual, visual, and audio data, LLMs enhance their ability to perform complex tasks accurately and contextually. Techniques such as retrieval-augmented generation and real-time web scraping allow these models to combine internal capabilities with external information, thereby improving planning and decision-making processes.

Utilization of Knowledge



Knowledge

textual visual audio others

issues

Edit Wrong & Outdated Knowledge

- information lag and deviation

Hallucination

- intrinsic and extrinsic hallucination

Spurious Bias

- shortcut learning and label bias

retrieval

solutions

Database and Knowledge Base Queries

- SQL, Google Knowledge Graph, Wikidata, PubMed, YAGO

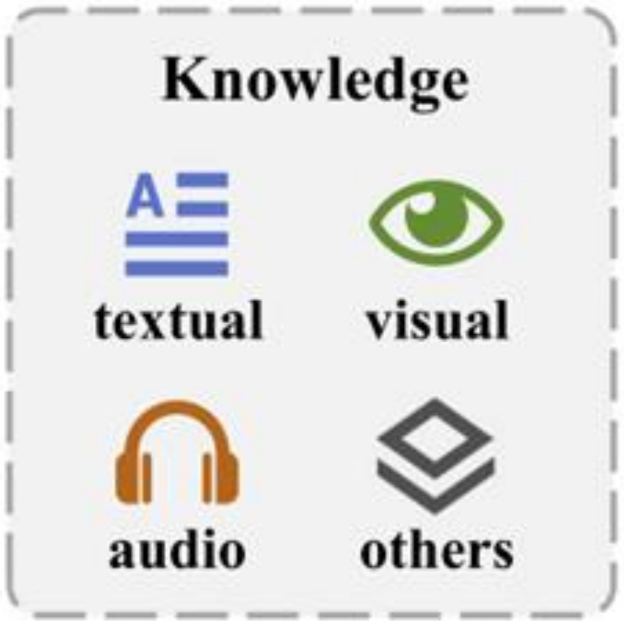
Web Scraping and API Calls

- web search, specific APIs (compiler, calculator, translator etc.)

Retrieval-Augmented Generation (RAG)

- retrieval mechanisms + generative models → context-rich responses

Utilization of Knowledge



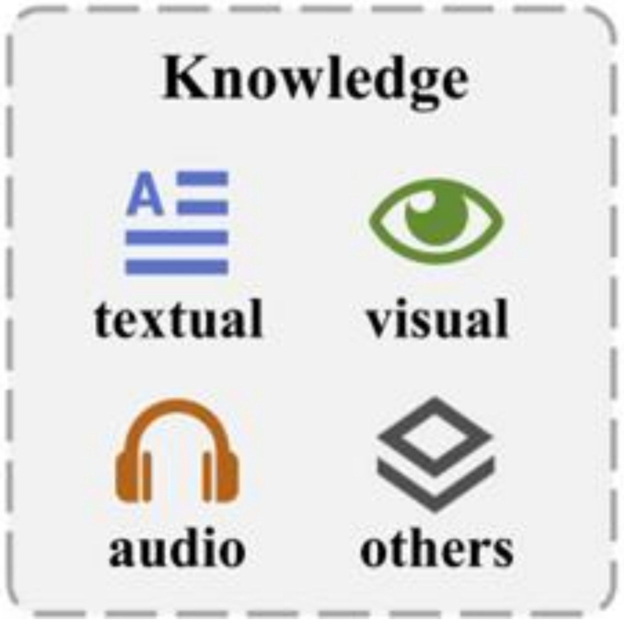
Textual knowledge is the backbone of LLMs, given their training on extensive text corpora. knowledge is vital for tasks such as natural language understanding, text generation, translation, and more. The formats of textual knowledge include natural language, embeddings, tokens, and tree structures. Natural language is the primary input and output format.

Utilization of Knowledge

Knowledge
textual
visual
audio
others

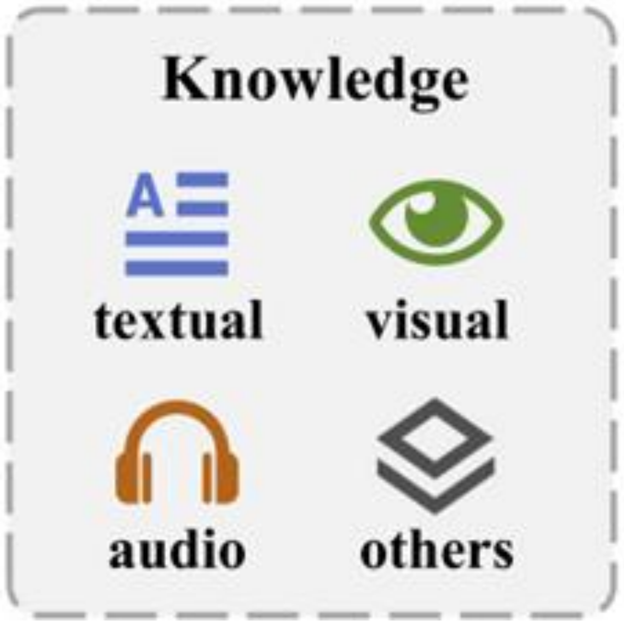
Visual knowledge is primarily represented through continuous embeddings generated by visual encoders, which are then integrated with textual information to facilitate multi-modal data understanding and reasoning. The representation of visual knowledge typically includes latent vector representations of images (e.g., visual Transformer encodings), object-centric encodings, and other forms, all processed alongside language information through standard self-attention mechanisms. LLM agents leverage these visual embeddings to achieve strong performance across various tasks, such as VQA, image captioning, and embodied reasoning.

Utilization of Knowledge



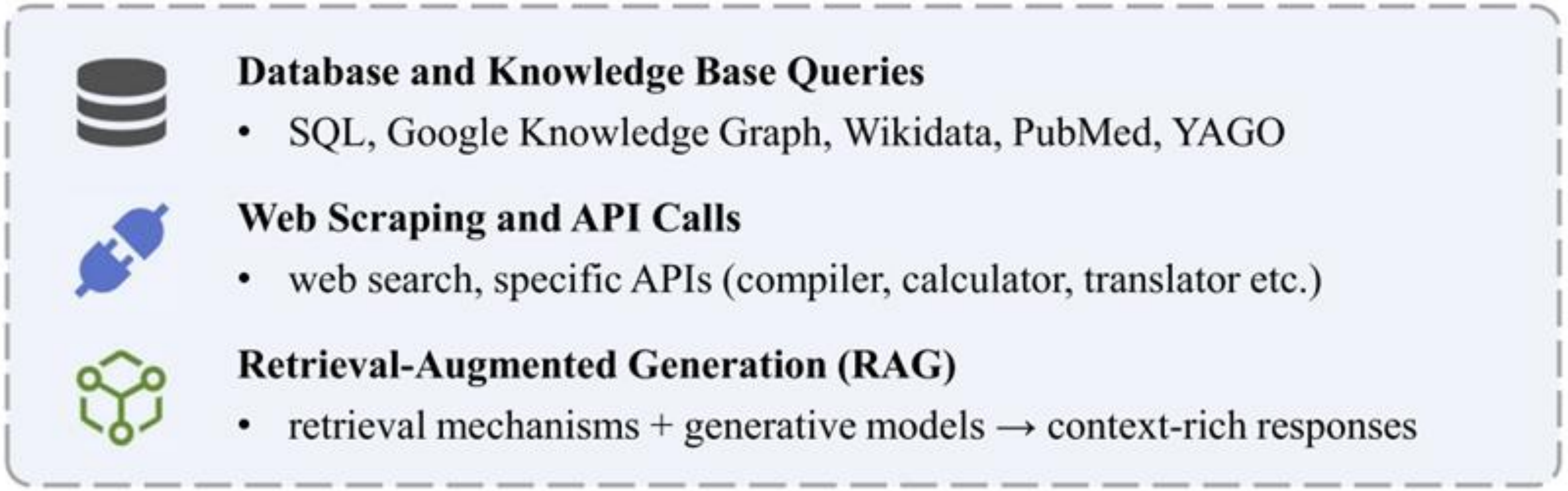
Audio knowledge encompasses speech and audio events, which can be represented through forms such as speech encoders and spectrogram images. When processing speech, LLM agents can discretize speech input via connection modules and embed it into a vector space shared with text.

Utilization of Knowledge



Other Knowledge. Beyond text, visual, and audio data, LLMs often need to utilize specialized knowledge from specific domains such as scientific research, medical information, or technical specifications. This enhances their ability to handle tasks that require deep domain expere.

Utilization of Knowledge



Database and Knowledge Base Queries

- SQL, Google Knowledge Graph, Wikidata, PubMed, YAGO

Web Scraping and API Calls

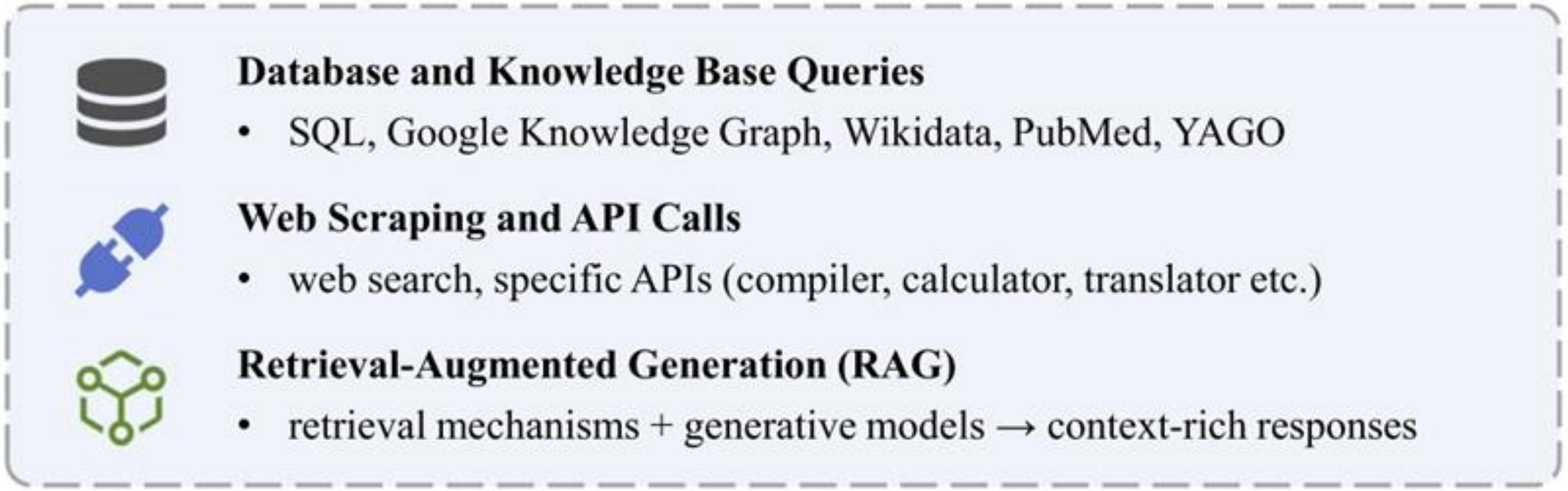
- web search, specific APIs (compiler, calculator, translator etc.)

Retrieval-Augmented Generation (RAG)

- retrieval mechanisms + generative models → context-rich responses

Database and Knowledge Base Queries. Database and knowledge base queries involve accessing structured data from repositories like Google Knowledge Graph, PubMed, and other domain-specific databases. These sources offer reliable and organized information that can be integrated with LLM outputs to enhance the accuracy and relevance of generated responses. A notable example of integrating external databases is the **ChatDB** system, which uses SQL queries to fetch relevant data logically, making it easier for agents to operate

Utilization of Knowledge



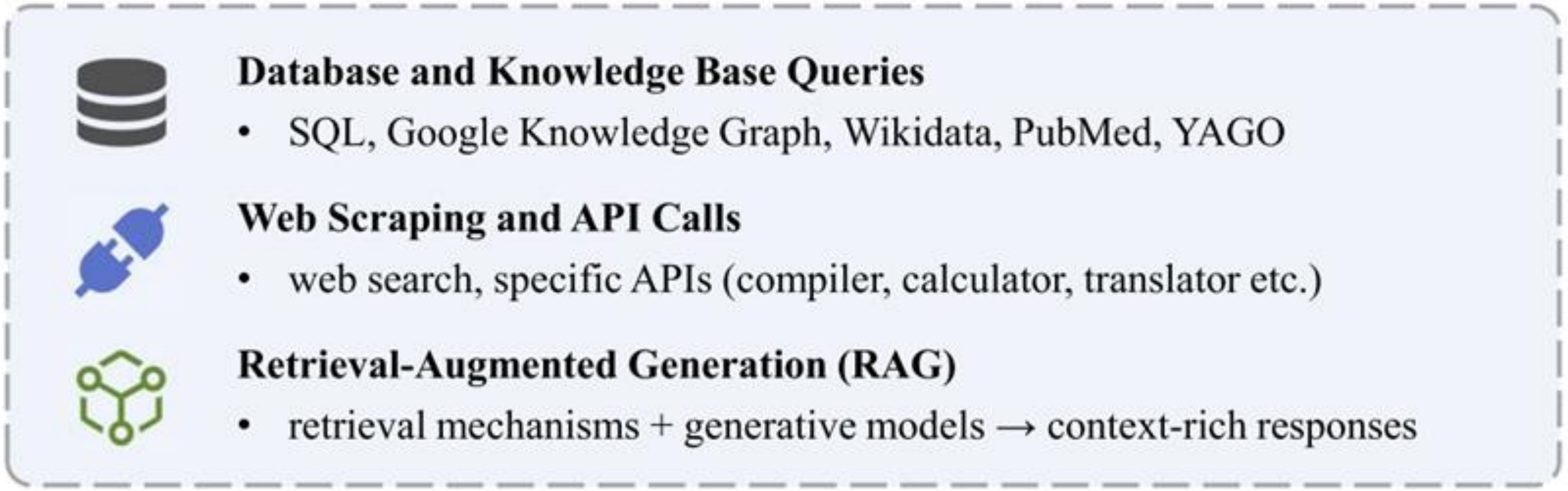
Database and Knowledge Base Queries
- SQL, Google Knowledge Graph, Wikidata, PubMed, YAGO

Web Scraping and API Calls
- web search, specific APIs (compiler, calculator, translator etc.)

Retrieval-Augmented Generation (RAG)
- retrieval mechanisms + generative models → context-rich responses

Web Scraping and API Calls. Web scraping and API calls allow LLM-based agents to collect real-time information from the internet. method is particularly useful for tasks requiring up-to-date data, such as news summarization or market analysis. Web scraping involves using automated tools to extract data from web pages, providing large amounts of data from diverse sources. API calls, on the other hand, involve querying APIs to fetch specific information, such as news articles, weather updates, or financial data

Utilization of Knowledge



Database and Knowledge Base Queries

- SQL, Google Knowledge Graph, Wikidata, PubMed, YAGO

Web Scraping and API Calls

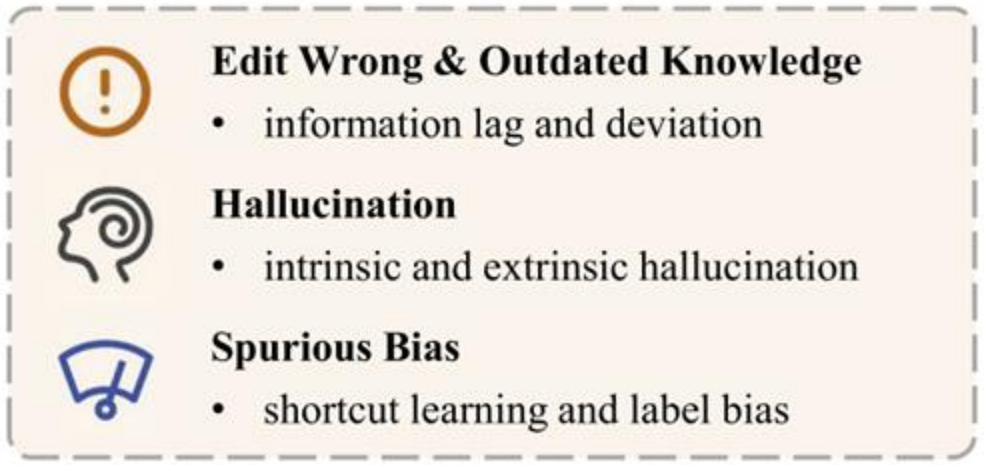
- web search, specific APIs (compiler, calculator, translator etc.)

Retrieval-Augmented Generation (RAG)

- retrieval mechanisms + generative models → context-rich responses

Retrieval-Augmented Generation (RAG). RAG models combine retrieval mechanisms with generative models to produce context-rich responses. approach is effective for open-domain question answering and conversational agents. The primary retrieval source is textual data, but it can be extended to semistructured data (e.g., PDFs), structured data, and content generated by LLMs themselves.

Utilization of Knowledge



Edit Wrong & Outdated Knowledge
- information lag and deviation

Hallucination
- intrinsic and extrinsic hallucination

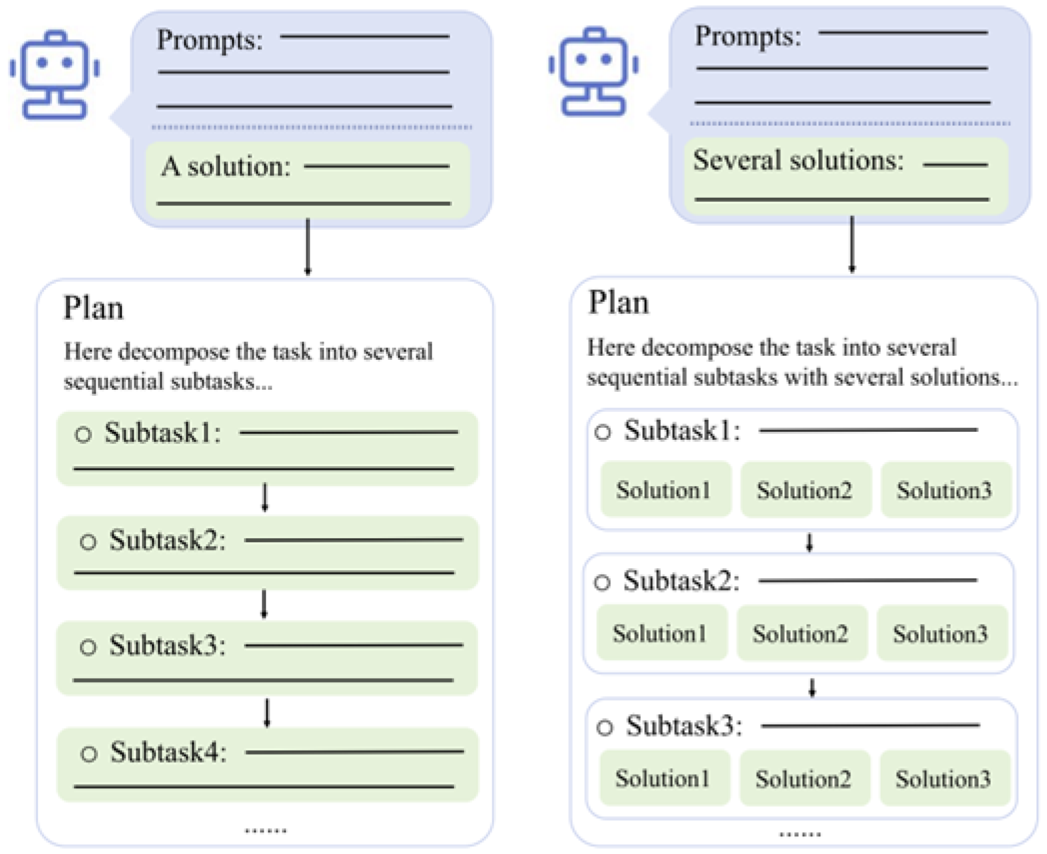
Spurious Bias
- shortcut learning and label bias

One of the primary challenges for LLM agents in knowledge extraction is ensuring the **timeliness and accuracy** of information. Therefore, developing efficient methods to incorporate new knowledge into existing LLMs to keep them up-to-date becomes paramount.

Hallucination refers to the phenomenon where LLM agents generate text that deviates from reality. One approach to reduce hallucinations is to integrate external knowledge bases and fact-checking systems to verify the accuracy of generated content (e.g., with RAGs).

To reduce **biases**, class imbalance, and compromissions caused by the training data, researchers propose rebalancing datasets, employing advanced sampling techniques, and developing new evaluation metrics to enhance model fairness and robustness

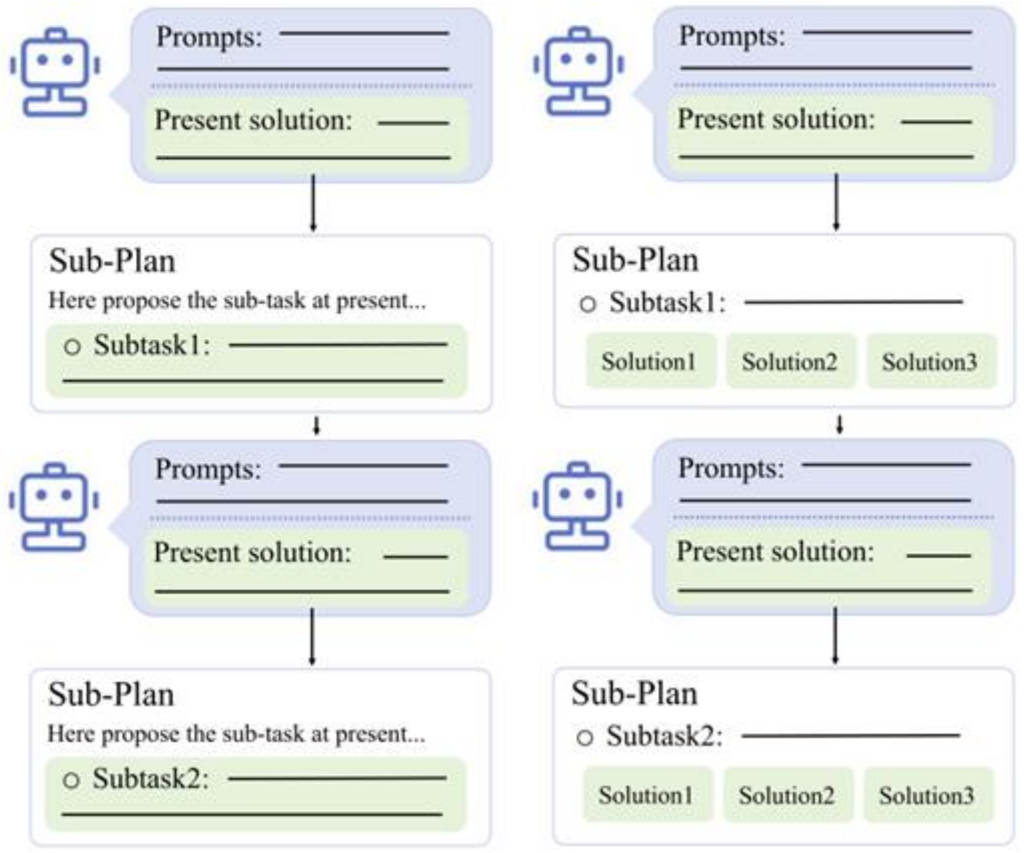
Reasoning and planning



One-Step Reasoning & Planning

One-Step Method. In this strategy, agents decompose a complex task into several sub-tasks through a single reasoning & planning process based on the current task directives. These sub-tasks are sequentially ordered, with each sub-task logically following the preceding one. LLM-based agents adhere to these steps to achieve the final objective.

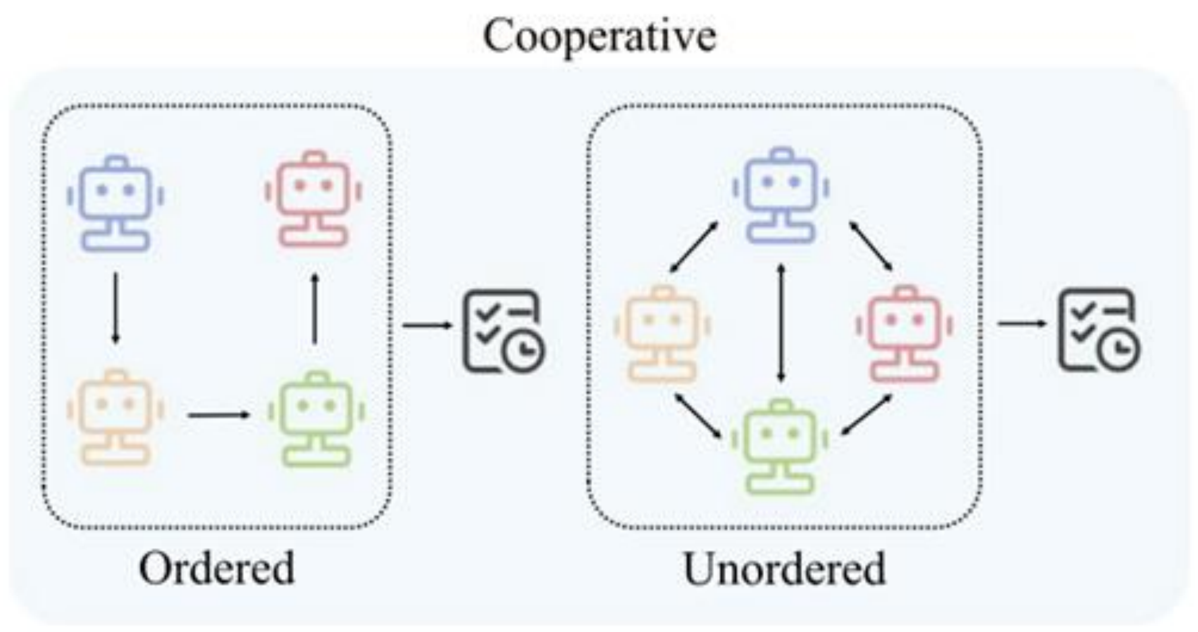
Reasoning and planning



Multi-Step Reasoning & Planning

Multi-step method. Unlike one-step reasoning, multep reasoning requires iterative invocation of LLMs for multiple reasoning cycles, where each cycle generates one or several incremental steps based on the current context while maintaining consistency with the overall objective.

Agent interaction Schemes

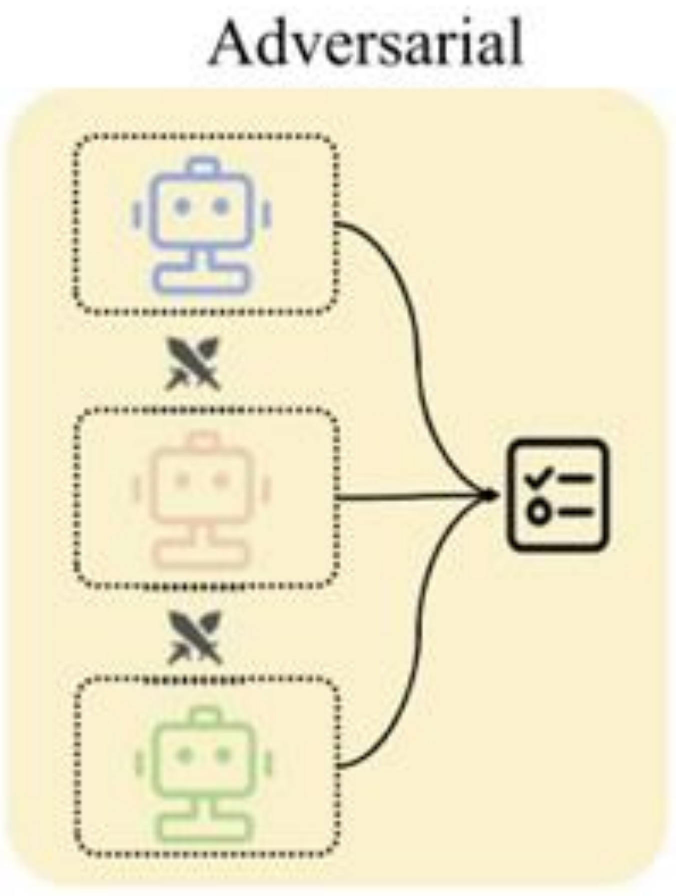


Cooperative. In cooperative interaction scenarios, agents work together to achieve a common goal. The basic process of cooperative MAS includes goal setting, task decomposition, information sharing, collaborative decision-making, and execution feedback.

Agents first set common goals based on task requirements, then decompose complex tasks into multiple subtasks assigned to different agents. The agents share information and jointly make decisions through communication and negotiation to reach a consensus.

During task execution, agents perform tasks based on their respective roles and provide feedback to adjust strategies and optimize the execution process.

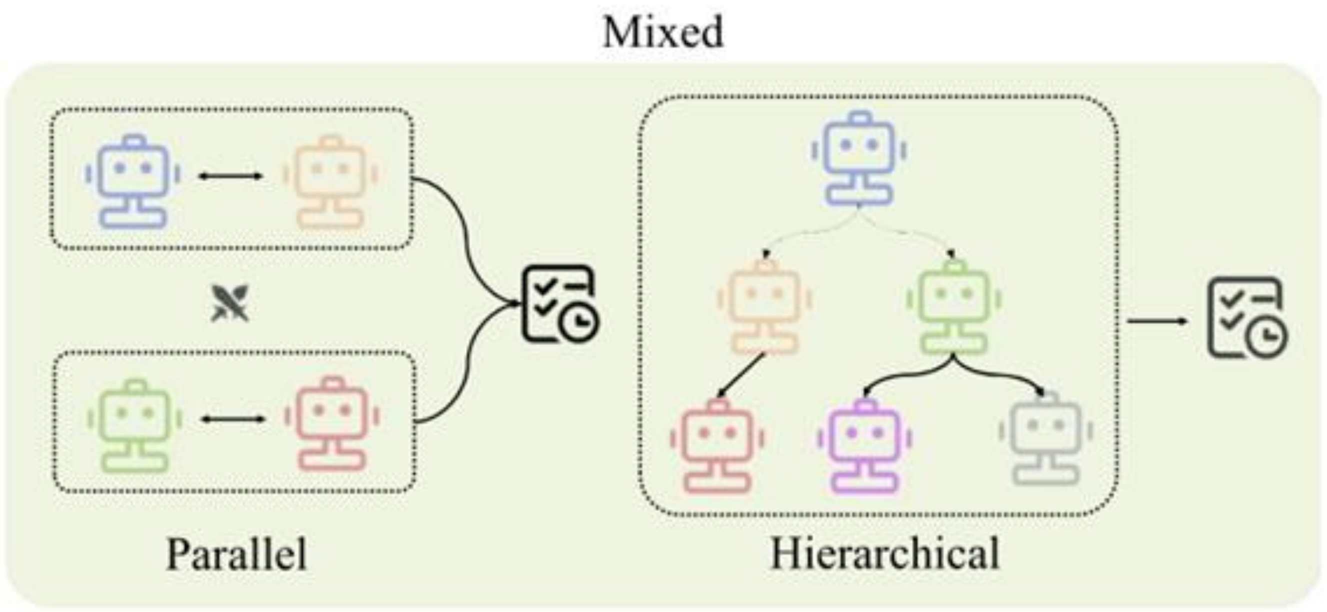
Agent interaction Schemes



Adversarial. In adversarial interaction scenarios, agents are in a competitive relationship, each pursuing the maximization of their own interests. The basic process includes goal setting, strategy formulation, interaction games, and result evaluation. Agents frst set goals to maximize their own interests and then formulate competitive strategies based on the behavior of their opponents. In the interaction game stage, agents implement strategies through interactions to strive for maximum benefts. Finally, agents evaluate the game results and adjust strategies to cope with future competition.

Example: ChatEval

Agent interaction Schemes

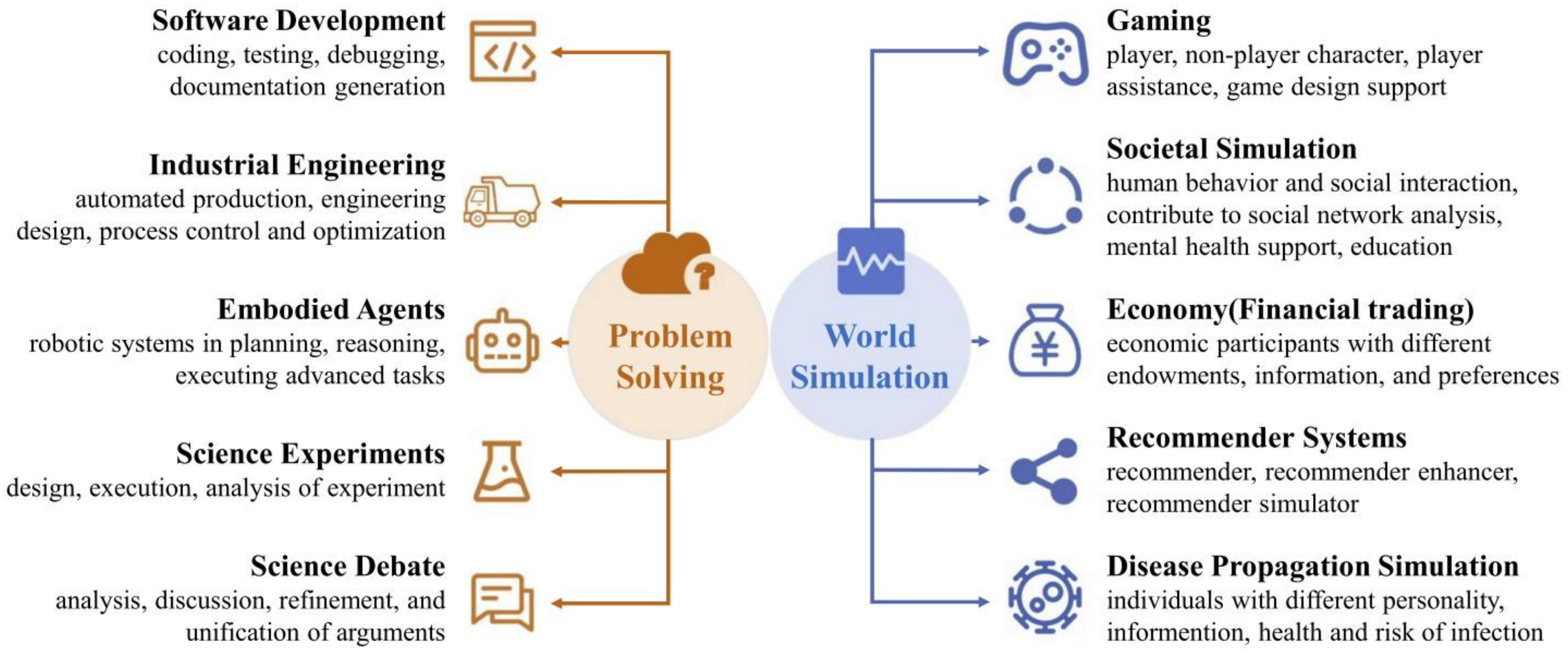


Mixed. interaction scenarios combine features of both cooperative and adversarial interactions, requiring agents to fnd a balance between cooperation and competition. type of interaction can be further subdivided into parallel and hierarchical forms.

Parallel: In parallel interactions, agents collaborate independently on separate tasks, sharing some information without interfering with each other. Then a form of competition is applied.

Hierarchical: the relationships among agents typically manifest as a tree structure. The parent node agents set global goals, decompose tasks, and assign them to child node agents. The child node agents execute specific tasks and provide feedback on the execution.

Application of LLM multi agent systems


Software Development
coding, testing, debugging, documentation generation
Industrial Engineering
automated production, engineering design, process control and optimization
Embodied Agents
robotic systems in planning, reasoning, executing advanced tasks
Science Experiments
design, execution, analysis of experiment
Science Debate
analysis, discussion, refinement, and unification of arguments
Problem Solving
World Simulation
Gaming
player, non-player character, player assistance, game design support
Societal Simulation
human behavior and social interaction, contribute to social network analysis, mental health support, education
Economy(Financial trading)
economic participants with different endowments, information, and preferences
Recommender Systems
recommender, recommender enhancer, recommender simulator
Disease Propagation Simulation
individuals with different personality, informention, health and risk of infection